(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 December 2003 (04.12.2003)

PCT

(10) International Publication Number
WO 03/100650 A1

(51) International Patent Classification[7]: G06F 15/177

(21) International Application Number: PCT/US03/15910

(22) International Filing Date: 21 May 2003 (21.05.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/152,532 21 May 2002 (21.05.2002) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 10/152,532 (CON)
Filed on 21 May 2002 (21.05.2002)

(71) Applicant (*for all designated States except US*): **WASHINGTON UNIVERSITY** [US/US]; One Brookings Drive, St. Louis, MO 63130 (US).

(72) Inventors; and

(75) Inventors/Applicants (*for US only*): **LOCKWOOD, John** [US/US]; 839 Jackson Avenue, St. Louis, MO 63130 (US). **LOUI, Ronald** [US/US]; 7920 Lafon Place, St. Louis, MO 63130 (US). **MOSCOLA, James** [US/US]; 1151-H Olive Lake Drive, St. Louis, MO 63132 (US). **PACHOS, Michael** [US/US]; 228 Newbury Street, #32, Boston, MA 02116 (US).

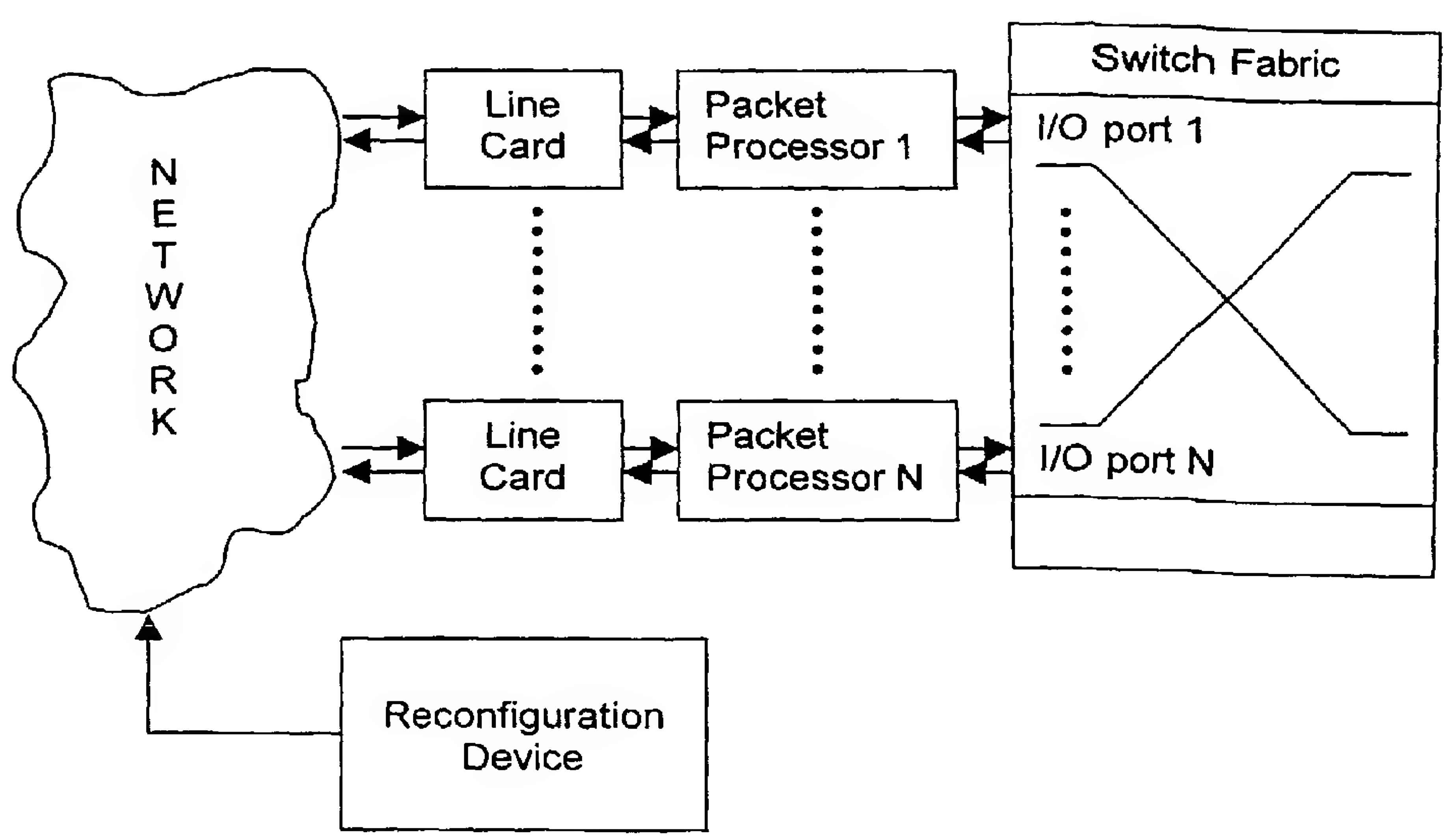
(74) Agents: **VOLK, Benjamin, L., Jr.** et al.; Thompson Coburn LLP, One US Bank Plaza, St. Louis, MO 63101 (US).

(81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: REPROGRAMMABLE HARDWARE FOR EXAMINING NETWORK STREAMING DATA TO DETECT REDEFINABLE PATTERNS AND DEFINE RESPONSIVE PROCESSING



(57) Abstract: A reprogrammable packet processing system for processing streams is disclosed. A reprogrammable data processor is programmed to determine whether a stream of data includes a string that matches a specific data pattern. If so, the data processor performs a specified action. The data processor is reprogrammable to search packets for the presence of different data patterns and/or perform different actions when a matching string is detected. A reconfiguration device receives input from a user specifying the data pattern and action, processes the input to generate the configuration information necessary to reprogram the data processor, and transmits the configuration information to the packet processor for reprogramming thereof.

WO 03/100650 A1

Published:
- *with international search report*
- *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHODS, SYSTEMS, AND DEVICES USING REPROGRAMMABLE HARDWARE FOR HIGH-SPEED PROCESSING OF STREAMING DATA TO FIND A REDEFINABLE PATTERN AND RESPOND THERETO

FIELD OF THE INVENTION

The present invention relates to high-speed processing of data, such as packets transmitted over computer networks. More specifically, the present invention relates to the processing of packet payloads to (1) detect whether a string is present in those payloads that matches a redefinable data pattern, and (2) perform a redefinable action upon detection thereof.

BACKGROUND OF THE INVENTION

It is highly desirable to possess the ability to monitor the content of packets transmitted over computer networks. Whether the motivation is to identify the transmission of data files containing material such as copyrighted audio, film/video, software, published articles and book content, to secure confidential data within a company's internal computer system, to detect and eliminate computer viruses, to identify and locate packet transmissions that may be part of a criminal conspiracy (such as e-mail traffic between two persons

planning a crime), or to monitor data transmissions of targeted entities, the ability to search packet payloads for strings that match a specified data pattern is a powerful tool in today's electronic information age. Further, the ability to modify the data stream permits the system to, among other things, filter data, reformat data, translate between languages, extract information, insert data, or to notify others regarding the content.

String matching and pattern matching have been the subject of extensive studies. In the past, software-based string matching techniques have been employed to determine whether a packet payload includes a data pattern. However, such software-based techniques are impractical for widespread use in computer networks because of the inherently slow packet processing speeds that result from software execution.

For example, U.S. Patent No. 5,319,776 issued to Hile et al. (the disclosure of which is hereby incorporated by reference) discloses a system wherein data in transit between a source medium and a destination medium is tested using a finite state machine capable of determining whether the data includes any strings that represent the signatures of known computer viruses. However, because the finite state machine of Hile is implemented in software, the Hile system is slow. As such, the Hile system is impractical for use as a network device capable of handling high-speed line rates such as OC-48 where the data rate approaches 2.5 gigabits per second. Furthermore, software-based techniques are traditionally and inherently orders of magnitude slower than a hardware-based technique.

Another software-based string matching technique is found in U.S. Patent No. 5,101,424 issued to Clayton et al. (the disclosure of which is hereby incorporated by reference). Clayton discloses a software-based AWK processor for monitoring text streams from a telephone switch. In Clayton, a data stream passing through a telephone switch is loaded into a text file. The Clayton system then (1) processes the content of the text file to determine if particular strings are found therein, and (2) takes a specified action upon finding a match. As with the Hile system described above, this software-based technique is too slow to be practical for use as a high-speed network device.

Furthermore, a software tool known in the art called SNORT was developed to scan Internet packets for combinations of headers and payloads that indicate whether a computer on a network has been compromised. This software program is an Open Source Network Intrusion Detection System that scans packets that arrive on a network interface. Usually, the packets arrive on a media like Ethernet. The program compares each packet with the data specified in a list of rules. If the fields in the header or parts of the payload match a rule, the program performs responsive tasks such as printing a message on a console, sending a notification message, or logging an event to a database. Details of SNORT are described on the SNORT homepage, http://www.snort.org/. As with the above-described systems, SNORT, by virtue of being implemented in software, suffers from slow processing speed with respect to both its matching tasks and its responsive tasks.

In an effort to improve the speed at which packet payloads are processed, systems have been designed with dedicated application specific integrated circuits (ASICs) that scan packet payloads for a particular string. While the implementation of payload scanning on an ASIC represented a great speed improvement over software-based techniques, such ASIC-based systems suffered from a tremendous flexibility problem. That is, ASIC-based payload processing devices are not able to change the search string against which packets are compared because a change in the search string necessitates the design of a new ASIC tailored for the new search string (and the replacement of the previous ASIC with the new ASIC). That is, the chip performing the string matching would have to be replaced every time the search string is changed. Such redesign and replacement efforts are tremendously time-consuming and costly, especially when such ASIC-based systems are in widespread use.

To avoid the slow processing speed of software-based pattern matching and the inflexibility of ASIC-based pattern matching, reprogrammable hardware, such as field programmable gate arrays (FPGAs), have been employed to carry out pattern matching. Such an FPGA-based technique is disclosed in Sidhu, R. and Prasanna, V., "Fast Regular Expression Matching using FPGAs", IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001), April 2001 and Sidhu, R. et al., "String Matching on Multicontext FPGAs Using Self-Reconfiguration", FPGA '99: Proceedings of the 1999 ACM/SIGDA

Seventh International Symposium on Field Programmable Gate Arrays, pp. 217-226, February 1999, the entire disclosures of which are hereby incorporated by reference.

The Sidhu papers disclose a technique for processing a user-specified data pattern to generate a non-deterministic finite automata (NFA) operable upon being programmed into a FPGA to determine whether data applied thereto includes a string that matches a data pattern. However, Sidhu fails to address how such a device can also be programmed to carry out a specified action, such as data modification, in the event a matching string is found in the data. Thus, while the Sidhu technique, in using an FPGA to perform pattern matching for a redefinable data pattern, provides high speed through hardware implementation and flexibility in redefining a data pattern through the reprogrammable aspects of the FPGA, the Sidhu technique fails to satisfy a need in the art for a device which not only detects a matching string, but also carries out a specified action upon the detection of a matching string.

Moreover, while the Sidhu technique is capable of scanning a data stream for the presence of any of a plurality of data patterns (where a match is found if P_1 or P_2 or … or P_n is found in the data stream – wherein P_i is the data pattern), the Sidhu technique is not capable of either identifying which data pattern(s) matched a string in the data stream or which string(s) in the data stream matched any of the data patterns.

Unsatisfied with the capabilities of the existing FPGA-based pattern matching techniques, the inventors herein have sought to design a packet processing system able to not only determine whether a packet's payload includes a string that matches a data pattern in a manner that is both high-speed and flexible, but also perform specified actions when a matching string is found in a packet's payload.

An early attempt by one of the inventors herein at designing such a system is referred to herein as the "Hello World Application". See Lockwood, John and Lim, David, *Hello, World: A Simple Application for the Field Programmable Port Extender (FPX)*, Washington University Tech Report WUCS-00-12, July 11,2000 (the disclosure of which is hereby incorporated by reference). In the Hello World Application, a platform using reprogrammable hardware for carrying out packet processing, known as the Washington University

Field-Programmable Port Extender (FPX) (see Figure 10), was programmed with a state machine and a word counter designed to (1) identify when a string comprised of the word "HELL" followed by the word "O***" (wherein each * represents white space) was present in the first two words of a packet payload, and (2) when that string is found as the first two words of a packet payload, replace the word "O***" with the word "O*WO" and append the words "RLD." and "****" as the next two words of the packet payload. The reprogrammable hardware used by the FPX was a field programmable gate array (FPGA). The Hello World Application thus operated to modify a packet with "HELLO" in the payload by replacing "HELLO" with "HELLO WORLD".

While the successful operation of the Hello World Application illustrated to the inventors herein that the implementation of a circuit in reprogrammable hardware capable of carrying out exact matching and string replacement was feasible, the Hello World Application was not accompanied by any device capable of taking full advantage of the application's reprogrammable aspects. That is, while the FPGA programmed to carry out the Hello World Application was potentially reprogrammable, no technique had been developed which would allow the FPGA to be reprogrammed in an automated and efficient manner to scan packets for a search string other than "HELLO", or to replace the matching string with a replacement string other than "HELLO WORLD". The present invention addresses a streamlined process for reprogramming a packet processor to scan packets for different redefinable strings and carry out different redefinable actions upon packets that include a matching string. Toward this end, the present invention utilizes regular expressions and awk capabilities to create a reprogrammable hardware-based packet processor having expanded pattern matching abilities and the ability to take a specified action upon detection of a matching string.

Regular expressions are well-known tools for defining conditional strings. A regular expression may match several different strings. By incorporating various regular expression operators in a pattern definition, such a pattern definition may encompass a plurality of different strings. For example, the regular expression operator ".*" means "any number of any characters". Thus, the regular expression "c.*t" defines a data pattern that encompasses strings such as "cat", "coat", "chevrolet", and "cold is the opposite of hot". Another example of a regular expression operator is "*"

which means "zero or more of the preceding expression". Thus, the regular expression "a*b" defines a data pattern that encompasses strings such as "ab", "aab", and "aaab", but not "acb" or "aacb". Further, the regular expression "(ab)*c" encompasses strings such as "abc", "ababc", "abababc", but not "abac" or "abdc". Further still, regular expression operators can be combined for additional flexibility in defining patterns. For example, the regular expression "(ab)*c.*z" would encompass strings such as the alphabet "abcdefghijklmnopqrstuvwxyz", "ababcz", "ababcqsrz", and "abcz", but not "abacz", "ababc" or "ababacxvhgfjz".

As regular expressions are well-known in the art, it is unnecessary to list all possible regular expression operators (for example, there is also an OR operator "|" which for "(a|b)" means any string having "a" or "b") and combinations of regular expression operators. What is to be understood from the background material described above is that regular expressions provide a powerful tool for defining a data pattern that encompasses strings of interest to a user of the invention.

Further, awk is a well-known pattern matching program. Awk is widely used to search data for a particular occurrence of a pattern and then perform a specified operation on the data. Regular expressions can be used to define the pattern against which the data is compared. Upon locating a string encompassed by the pattern defined by the regular expression, awk allows for a variety of specified operations to be performed on the data. Examples of specified operations include simple substitution (replacement), back substitution, guarded substitution, and record separation. These examples are illustrative only and do not encompass the full range of operations available in awk for processing data.

As a further improvement to the Hello World Application, the present invention provides users with the ability to flexibly define a search pattern that encompasses a plurality of different search strings and perform a variety of awk-like modification operations on packets. These features are incorporated into the reprogrammable hardware of the present invention to produce a packet processor having a combination of flexibility and speed that was previously unknown.

SUMMARY OF THE INVENTION

Accordingly, disclosed herein is a reprogrammable data processing system for a stream of data.

One component of the system comprises a reprogrammable data processor for receiving a stream of data and processing that data stream through a programmable logic device (PLD) programmed with a data processing module that is operable to (1) determine whether a string that matches a redefinable data pattern is present in the data stream, and (2) perform a redefinable action in the event such a matching string is found. The data pattern may be defined by a regular expression, and as such, may encompass a plurality of different strings. Additionally, the data stream processed by the data processor may be a stream of data packets transmitted over a computer network, in which case the data processor is a packet processor and the data processing module is a packet processing module. Also, such a packet processing module may be operable to determine whether the payloads of received packets include a string that matches the data pattern. The PLD is preferably a field programmable gate array (FPGA).

Examples of redefinable actions that can be performed by the data processor upon detection of a matching string are modification operations (eg, awk tasks such as string replacement, back substitution, etc.), drop operations, notification operations (wherein an interested party is informed that a match has occurred – the notification can encompass varying levels of detail (a copy of the packet that includes the matching string, a notice of the data pattern that matched a string, a notice of the string that matched a data pattern)), and record-keeping/statistical updates (wherein data is gathered as to the content of the data stream).

Another component of the system is a device for generating configuration information operable to program a PLD with a data processing module, the device comprising: (1) an input operable to receive a data pattern and an action command from a user; (2) a compiler operable to generate configuration information at least in part from the received data pattern and action command (the configuration information defining a data processing module operable to determine whether a data stream applied thereto includes a string that matches the received data pattern), wherein the configuration information is operable to program the PLD with the data processing module. A transmitter may be used to communicate the configuration

information from the compiler to the PLD to thereby program the data processing module into the PLD.

The compiler preferably includes a lexical analyzer generator which automates the design of the data processing module. The lexical analyzer generator processes the received data pattern to create a logical representation of a pattern matching state machine at least partially therefrom. The pattern matching state machine carries out the task of determining whether a data stream includes a string that matches the received data pattern. The pattern matching state machine at least partially defines the data processing module.

Because its tasks are carried out in hardware, the data processor of the present invention is capable of operating a network line speeds. Further, because of the device that generates the configuration information used to program the data processor, the data processing system of the present invention is easily reprogrammed to search packets for additional or different data patterns by simply providing the additional or different data pattern thereto, and is also easily reprogrammed to carry out additional or different actions in response to detecting a matching string. Once such input is supplied by a user, the compiler generates the necessary configuration information to carry out the reprogramming and the transmitter communicates that information to the data processor, possibly via a computer network. Not only is the data processor reprogrammable to search packets for different data patterns, but it is also reprogrammable by the same techniques to carry out different packet modification operations. Accordingly, the speed and flexibility of the present invention is unrivaled in the prior art.

Because of this speed and flexibility, the potential applications for the present invention are wide-ranging. For example, the present invention can be used for virus detection. The data pattern with which a packet processor of the present invention is keyed may be a data pattern that encompasses a known computer virus. Thus, the present invention may be used to detect (and eliminate through the modification operation) any known computer viruses that are present in a packet transmission.

Also, the present invention can be used to police copyrights. The packet processor can be keyed with a data pattern that will reliably detect when a party's copyrighted material is transmitted

over a network. For example, copyrighted songs, motion pictures, and images are often transmitted over the web via audio files, video files, and image files. By properly designing a data pattern that will detect when such works are present in packet traffic, a practitioner of the present invention can utilize the packet processor to detect the transmission of such copyrighted works and take appropriate action upon detection.

Further still, the present invention can be used to protect against the dissemination of trade secrets and confidential documents. A company having trade secrets and/or confidential documents stored on its internal computer system can utilize the present invention to prevent the unauthorized transmission of such information outside the company's internal network. The company's network firewall can use a packet processor that is keyed to detect and drop any unauthorized packets that are found to include a string that matches a data pattern that encompasses that company's trade secrets and/or confidential information. A company has a wide range of options for flagging their confidential/trade secret information, from adding electronic watermarks to such information (wherein the data processor is keyed by the watermark) to designing a separate data pattern for each confidential/trade secret document/file that will reliably detect when that document/file is transmitted.

Further still, the present invention can be utilized by governmental investigatory agencies to monitor data transmissions of targeted entities over a computer network. The packet processor can be keyed with a data pattern that encompasses keywords of interest and variations thereof. For example, certain words related to explosives (i.e., TNT, etc.), crimes (i.e., kill, rob, etc.), and/or wanted individuals (i.e., known terrorists, fugitives, etc.) can be keyed into the packet processor. Once so configured, the packet processor can detect whether those keywords (or variations) are present in a packet stream, and upon detection take appropriate action (e.g.., notify an interested governmental agency, or redirect the data for further automated processing).

Yet another example of an application for the present invention is as a language translator. The packet processor's search and replace capabilities can be used to detect when a word in a first language is present in a packet, and upon detection, replace that word with its translation into a second language. For example, the

packet processor can be used to replace the word "friend" when detected in a packet with its Spanish translation "amigo". Taking advantage of the fact that the packet processor of the present invention possesses the capability of searching packets for a plurality of different data patterns, the present invention can be used as a large scale translation device wherein the packet processor is keyed with a large language A-to-language B dictionary. Further still, it is possible that a practitioner of the present invention can develop data patterns that not only take into account word-for-word translations, but also will account for grammatical issues (for example, to reconcile the English method of a noun preceded by an adjective with the Spanish method of a noun followed by an adjective).

Further still, the present invention can be used to monitor/filter packet traffic for offensive content. For example, a parent may wish to use the packet processor of the present invention to prevent a child from receiving profane or pornographic material over the Internet. By keying the data processor to search for and delete profanities or potentially pornographic material, a parent can prevent such offensive material from reaching their home computer.

Yet another potential application is as an encryption/ decryption device. The packet processor can be designed to replace various words or letters with replacement codes to thereby encrypt packets designed for the network. On the receiving end, a packet processor can be equipped to decrypt the encrypted packets by replacing the replacement codes with the original data.

These are but a few of the potential uses of the present invention. Those of ordinary skill in the art will readily recognize additional uses for the present invention, and as such, the scope of the present invention should not be limited to the above-described applications which are merely illustrative of the wide range of usefulness possessed by the present invention. The full scope of the present invention can be determined upon review of the description below and the attached claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1(a) is an overview of the packet processing system of the present invention;

Figure 1(b) is an illustration of an example of the search and replace capabilities of the packet processor of the present invention;

Figure 2 is an overview of how the packet processing system of the present invention may be implemented in a high-speed computer network;

Figure 3 is an overview of a module programmed into a PLD that is operable to provide packet processing capabilities;

Figure 4 is an overview of a module programmed into a PLD that is operable to provide packet processing capabilities, wherein the module is capable of search and replace functionality for more than one data pattern;

Figure 5 is a diagram of the search and replace logic operable to determine whether incoming data includes a string that matches a specified data pattern and replace any matching string with a replacement string;

Figure 6 is an illustration of a packet and the content of the words comprising that packet;

Figures 7(a) and 7(b) are flowcharts illustrating how the controller determines the starting position and ending position of a matching string;

Figure 8 is a flowchart illustrating how the controller controls the outputting of data, including the replacement of a matching string with a replacement string;

Figure 9 is a flowchart illustrating the operation of the replacement buffer;

Figure 10 is a flowchart illustrating the operation of the byte-to-word converter;

Figure 11 is a flowchart illustrating how the controller accounts for changes in the byte length of modified packets;

Figure 12 is a diagram of the Field-Programmable Port Extender (FPX) platform;

Figure 13 is an overview of the reconfiguration device of the present invention;

Figure 14 is a flowchart illustrating the operation of the compiler

Figure 15 is a diagram of an implementation of the matching path of the search and replace logic wherein multiple pattern matching state machines operate in parallel; and

Figure 16 is a flowchart illustrating how the controller controls the outputting of data wherein the data modification operation is a back substitution operation.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

An overview of the packet processing system of the present invention is shown in Figures 1(a) and 1(b). The packet processor operates to receive data packets transmitted over a computer network and scan those packets to determine whether they include a string that matches a specified data pattern. If the packet is found to include a matching string, the packet processor performs a specified action such as data modification (e.g. string replacement), a packet drop, a notification, or some other action. For example, the packet processor may be configured to determine whether a packet includes the string "recieved", and if that string is found, modify the packet by replacing each instance of "recieved" with the properly-spelled replacement string "received". Or, the packet processor may be configured to determine whether a packet includes a string indicative of a computer virus, and if such a string is found, drop the packet. Also, the packet processor may be configured to send a notification packet or a notification signal to another device if a matching string is found. This list of actions that the packet processor may perform upon detection of a matching string is illustrative only, and the present invention may utilize any of a variety of actions responsive to match detections.

An important feature of the packet processor is that it is reprogrammable to scan packets for different data patterns and/or carry out different specified actions. A programmable logic device (PLD) resident on the packet processor is programmed with a module operable to provide pattern matching functionality and, if a match is found, perform a specified action. By reconfiguring the PLD, the packet processor can be reprogrammed with new modules operable to scan packets for different data patterns and/or carry out different actions when matches are found. Because the packet processor relies on hardware to perform pattern matching, it is capable of scanning received packets at network line speeds. Thus, the packet processor can be used as a network device which processes streaming data traveling at network rates such as OC-48.

To reprogram the packet processor, the reconfiguration device transmits configuration information over the network to the packet processor. The configuration information defines the module that is to be programmed into the PLD. After receiving the configuration information over the network, the packet processor reconfigures the PLD in accordance with the received configuration information.

The reconfiguration device generates the configuration information from user input. Preferably, this input includes the data pattern against which the data packets are to be compared and an action command that specifies the action to be taken upon the detection of a matching string. Once in receipt of this input from the user, the reconfiguration device generates configuration information therefrom that defines a module keyed to the data pattern and action command received from the user.

Figure 1(b) illustrates how the packet processor operates to perform a simple search and replace task. In the example of Figure 1(b), the module programmed into the packet processor is tuned with the data pattern "U.*S.*A" which means a U, followed by any number of any characters, followed by an S, followed by any number of any characters, followed by an A. When a string matching that pattern is found in a packet, the module is also keyed to replace the matching string with the replacement string "United States". Thus, when a packet having a payload portion that includes the string "I live in the USA" is received and processed by the packet processor, that packet will be modified so that the payload portion of the outputted packet includes the string "I live in the United States" (the string "USA" will be detected and replaced with "United States").

Figure 2 illustrates the packet processor's use as a network device. In a preferred implementation, the packet processor can be used as an interface between a NxN packet switch and the line cards that carry data traffic to and from the packet switch. In the event it is desired that the packet processor be reprogrammed to scan packets for a different data pattern (or another data pattern) or carry out a different action (or another action) when matches are found, the reconfiguration device generates the necessary configuration information and transmits that information over the network to the packet processor for reprogramming thereof.

Figure 3 is an overview of the preferred packet processing module 100 that is programmed into the PLD. Incoming data is first

processed by a protocol wrapper 102. On the incoming side, the protocol wrapper 102 operates to process various header information in the incoming packets and provide control information to the search and replace logic (SRL) 104 that allows SRL 104 to delineate the different portions of the packet. The control information preferably provided by the protocol wrapper 102 to SRL 104 is a start of frame (SOF) indicator that identifies the first word of a packet, a DATA_EN signal that is asserted when subsequent words of the packet are passed to SRL 104, a start of datagram (SOD) signal that identifies the first word of the UDP header, and an end of frame (EOF) signal that identifies the last word of the packet's payload.

Figure 6 depicts the various components and word positions of a data packet. The first word is the ATM header. The SOF signal is asserted with the arrival of the ATM header. Following the ATM header, there will be at least 5 words of IP header. The DATA_EN signal is asserted beginning with the first word of the IP header and remains asserted for all subsequent words of the packet. Following the IP header words, the first word of the UDP header is located at word position x. The SOD signal is asserted with the first word of the UDP header. The UDP header comprises two words, and at word position x+2, the payload portion of the packet begins. The EOF signal is asserted with the last word of the payload portion at word position x+n. Thus, the payload comprises some number L of words (L=n-2). The next two words at word positions x+n+1 and x+n+2 comprise the packet's trailer.

A preferred embodiment of the protocol wrapper includes a cell wrapper which verifies that the incoming packet arrived on the proper virtual circuit or flow identifier, a frame wrapper which segments incoming packets into cells and reassembles outgoing cells into packets, an IP wrapper which verifies that each incoming IP packet has the correct checksum and computes a new checksum and length for each outgoing IP packet, and a UDP wrapper which verifies that each incoming UDP packet has the correct checksum and length and computes a new checksum and length for each outgoing UDP packet.

It is preferred that pattern matching be performed only upon the payload portion of a packet. In such cases, SRL 104 uses the control information (SOF, DATA_EN, SOD, and EOF) from the protocol wrapper 102 to identify the words of the packet that comprise the

payload and perform pattern matching only upon those words. However, this need not be the case.

Also, additional wrappers, such as a TCP wrapper may be included if desired by a practitioner of the invention. A TCP wrapper would read and compute checksums so that incoming packets are assembled into a continuous stream of data that is identical to the data stream that was transmitted. The TCP wrapper would drop data arriving in multiple copies of the same packet and reorder packets that arrive out of sequence.

The design and operation of the protocol wrappers is well known in the art. See, for example, F. Braun, J.W. Lockwood, and M. Waldvogel, "Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware", Washington University Technical Report WU-CS-01-10, Washington University in St. Louis, Dept. of Computer Science, June 2001, the disclosure of which is incorporated herein by reference.

Received data packets arrive at SRL as a stream of 32 bit words. Also, as stated, SRL 104 will receive 4 bits of control information with each word. SRL 104 is tuned with a data pattern from the reconfiguration device and operates to determine whether a string encompassed by that data pattern is present in the incoming word stream. SRL 104 is also tuned with a replacement string to carry out a string modification operation when a matching string is found in the incoming word stream. Examples of modification operations that the SRL 104 may carry out are any awk-based modification command, including straight substitution (replacing a matching string in the packet with a replacement string), back substitution (replacing a matching string in the packet with a replacement string, wherein the replacement string includes the actual string found in the packet that caused the match), and guarded substitution (adding or removing a string from a packet that exists in the packet either prior to or subsequent to the string in the packet that caused the match).

The module 100 may include a plurality N of SRLs 104 daisy-chained together, as shown in Figure 4. Each SRL can be keyed with a different data pattern and a different modification command. By allowing the packet processor of the present invention to scan packets for more than one data pattern, the capabilities of the packet processor are greatly enhanced. For example, if three

different computer viruses are known to be circulating in a computer network, the module 100 can include 3 different SRLs 104, each keyed with a data pattern designed for a different one of the computer viruses. When any of the SRLs detects a string in a packet matching its pattern, the SRL can remove the virus from the packet.

A schematic diagram of the SRL 104 is shown in Figure 5 along with a table that lists pertinent signals in the circuit. The operation of the SRL 104 is divided between a matching path 110, a data path 112, and a controller 114. The matching path 110 determines whether the incoming data stream includes a string that matches the specified data pattern. The data path 112 outputs the incoming data stream, and, if necessary, modifies that data stream in accordance with the specified modification operation. The controller 114 uses the control bits from the protocol wrapper and the control signals it receives from the matching path to coordinate and control the operation of both the matching path 110 and the data path 112.

As stated, the main task of the matching path 110 is to determine whether an input stream includes a string that matches the specified data pattern. The matching buffer (MB) receives a 35 bit streaming signal (S1) from the protocol wrapper. 32 bits will be a word of the packet, and 3 bits will be the SOF indicator, the SOD indicator, and the EOF indicator. Preferably, the incoming word is stored as the upper 32 bits in the matching buffer (MB) at the address identified by the DATA_WR_ADD signal (CS1) coming from the controller 114 and the control bits as the lower 3 bits. If the matching buffer (MB) is full, the controller asserts the CONGESTION signal (CS10) that notifies the protocol wrapper to stop sending data on S1. The MB will output a word buffered therein at the address specified by the MB_RD_ADD signal (CS2) coming from controller 114. The upper 32 bits (the word of the incoming packet) outputted from the MB (S2) are then be passed to word-to-byte converter 1 (WBC1). The lower 3 bits (the control bits for the word) are passed to the controller (S3) so that controller can decide how to process the 32 bit word corresponding thereto.

WBC1 operates to convert an incoming stream of words into an outgoing stream of bytes. WBC1 is preferably a multiplexor having 4 input groups of 8 bits a piece. Each input group will be one byte of the 32 bit word outputted from the MB. The WBC1_SELECT signal (CS3)

from the controller identifies which byte of the word is passed to the output of WBC1 (S4).

The output of WBC1 is received by the Regular Expression Finite State Machine (REFSM). The REFSM is a pattern matching state machine that processes an incoming byte stream to determine whether that byte stream includes a string that matches the data pattern with which it is keyed. Preferably, the pattern matching state machine of REFSM is implemented as a deterministic finite automaton (DFA). The REFSM processes the byte coming from WBC1 when the controller asserts the REFSM_ENABLE signal (CS4). Preferably, the controller asserts REFSM_ENABLE only when a byte of the packet's payload is present in S4.

As it processes bytes, the REFSM will produce an output signal REFSM_STATUS indicative of how the current byte being processed matches or doesn't match the data pattern with which it is keyed. REFSM_STATUS may indicate either a RUNNING state (a possible match), a RESETTING state (no match), or an ACCEPTING state (a matching string has been found). REFSM_STATUS will identify both the current state and next state of the REFSM, which depends upon the current byte being processed by the REFSM. If the REFSM processes a byte and determines that the byte is part of a string that may possibly match the data pattern (i.e., the string seen by the REFSM is "abc", the current byte is "d" and the data pattern is "abcde"), the current and next states of REFSM_STATUS will be RUNNING. If the REFSM processes a byte of a string that is a full match of the data pattern (the "e" byte has now been received), next state identified by REFSM_STATUS will be ACCEPTING. However, when the matching assignment is a longest match assignment, it must be noted that the ACCEPTING state does not necessarily mean that the REFSM's pattern matching tasks are complete. Depending on how the data pattern is defined, subsequent bytes to be processed by the REFSM may also match the data pattern. For example, if the data pattern as expressed in RE format is "abc*" (meaning that the data pattern is an "a" followed by a "b" followed by one or more "c"'s), once the REFSM has processed a byte stream of "abc", a match will have occurred, and the REFSM will be in the ACCEPTING state. However, if the next byte is also a "c", then the string "abcc" will also match the data pattern. As such, when in the ACCEPTING state, the REFSM will have to remain on alert for subsequent bytes that will continue the match. The REFSM will not

know that a full match has occurred until it receives and processes a byte that is not a "c".

REFSM_STATUS will identify when a full match when its current state is ACCEPTING and its next state is RESETTING (meaning that the current byte caused the match to fail and the previous byte was thus a full match). In the above example (where the RE is "abc*"), when the input string is "abcccd", the REFSM will begin RUNNING after processing the "a", will begin ACCEPTING when the first "c" is processed, and will MATCH when the final "c" is processed and the subsequent "d" causes the state machine to RESET.

The REFSM will provide the REFSM_STATUS signal (CS5) to the controller 114 to inform the controller both its current state and its next state (which will depend on the current byte). The controller will process the REFSM_STATUS signal to determine the MB word address and byte position in that word of the first and last bytes of a full match. Figures 7(a) and 7(b) illustrate how the controller controls the operation of the matching path 110 in this respect.

Steps 7.1-7.7 deal with processing the control bits associated with the words in the matching buffer (MB) to identify the words in MB that comprise a packet's payload. At step 7.1, the MB_RD_ADD signal is set equal to 1. The controller then asserts MB_RD_ADD (step 7.2) and the word stored in MB at that address is outputted as S2 while the control bits associated with that word is outputted as S3. At step 7.3, the controller checks whether the SOF bit is asserted. If it is, then the controller knows that the word stored in MB at MB_RD_ADD is the packet's ATM header. Referring back to Figure 6, it is known that the UDP header may possibly begin at word position 7 (if the IP header is 5 words long). Thus, when it is desired that the REFSM only process the packet's payload, the controller will adjust MB_RD_ADD to begin searching for the UDP header (once the first word of the UDP header is found, it is known that the first word of the payload will be two word positions later). Thus, at step 7.4, the controller sets MB_RD_ADD equal to MB_RD_ADD+6 and loops back to step 7.2.

Thereafter, on the next pass, the controller will arrive at step 7.5 and check the SOD bit associated with the word stored in MB at the location identified by the newly adjusted MB_RD_ADD signal. If the SOD bit is asserted, then the controller knows that the word

stored two addresses later in MB is the first word of the payload. Thus, if SOD is high, the controller sets MB_RD_ADD equal to MB_RD_ADD+2 (step 7.7) and begins the pattern matching process at step 7.8. If the SOD bit is not high, then the controller increments MB_RD_ADD until a word is found where the SOD bit is high (step 7.6).

Starting at 7.8, the pattern matching process begins. The first word of the packet's payload is outputted from the MB and the REFSM_ENABLE signal is asserted. Next, at step 7.9, the parameter BYTE_POS_IN is set to 1. This parameter is used to identify the byte of the word in S2 that is passed to WBC1's output. The controller asserts WBC1_SELECT = BYTE_POS_IN to thereby pass the first byte of the current word to the REFSM (step 7.10).

The REFSM then processes that byte, and asserts the REFSM_STATUS signal accordingly. The controller will read this signal (step 7.11). Next, at step 7.12, the controller checks whether the REFSM_STATUS signal indicates that the current state of the REFSM is the RUNNING state. If the current state is RUNNING, the controller proceeds to step 7.13 and stores both MB_RD_ADD and BYTE_POS_IN as the parameter BACK_PTR. From there, the controller proceeds to step 7.20 where it begins the process of finding the next byte to process. The parameter BACK_PTR will be a running representation of the current byte processed by the REFSM until the REFSM's current state is RUNNING, at which time the value of BACK_PTR is frozen. Due to the nature of flip-flops in the REFSM, REFSM_STATUS will not identify the current state as RUNNING until the second byte of a possible match is received. Thus, when REFSM_STATUS identifies a current state of RUNNING and BACK_PTR is frozen, the word and byte position identified by BACK_PTR will be the first byte of a possibly matching string.

If at step 7.12, the controller determines from REFSM_STATUS that the current state of the REFSM is RUNNING (meaning that the current byte may be part of a matching string), the controller will proceed to step 7.14 and check whether the next state identified by the REFSM_STATUS signal is RESETTING. If the next state is RESETTING, this means that the current byte has caused the partial match to fail. If the next state is not RESETTING, the controller (at step 7.15) checks whether the next state is ACCEPTING. If the next state is neither RESETTING nor ACCEPTING, this means that the next state is still RUNNING, in which case the controller proceeds to

step 7.20 to follow the process for obtaining the next byte of payload.

If at step 7.15, the controller determines that the next state is ACCEPTING, then this means that the REFSM has found a full match, but has not yet determined the full boundaries of the match. However, the controller does know that the word address and byte position of the current byte may be the word address and byte position of the last byte of the match. As such, the controller, at step 7.16, stores MB_RD_ADD and BYTE_POS_IN as the value ACCEPT_PTR. Then, at step 7.17, the controller notes that a match has occurred, and proceeds to step 7.20 to get the next byte.

As the next byte is processed and step 7.14 is once again reached, the controller will once again check whether the next state identified by REFSM_STATUS is RESETTING. If the next state is RESETTING, the controller proceeds to step 7.18 where it checks whether a match has been previously noted by the controller. If no match had been noted, the controller will determine that the string starting at the byte identified by BACK_PTR is not a match of the data pattern. Thus, the controller needs to set MB_RD_ADD and BYTE_POS_IN such that the REFSM will process the byte immediately after BACK_PTR, because the byte stored at that address needs to be checked to determine whether it may be the beginning of a matching string. The controller achieves this by setting MB_RD_ADD and BYTE_POS_IN equal to the values stored in BACK_PTR (step 7.19). From there, the controller proceeds to step 7.20 to get the next byte.

However, in the example where the controller had already noted that a match occurred at step 7.17, then, when the controller subsequently arrives at step 7.18, the controller will proceed to step 7.29. When step 7.29 is reached, this means that the full boundaries of a matching string have been processed by the REFSM. The current byte has caused the REFSM to determine that its next state is RESETTING. However, the previous byte (whose location is identified by ACCEPT_PTR) will be the last byte of the matching string. Also, the value of BACK_PTR will be the address of the first byte of the matching string. Thus, the controller will know the address of the first and last bytes of the longest matching string in the packet's payload. At step 7.29, the controller will store the value of BACK_PTR in FIFO A as START_ADD (CS6 in Figure 5). Also, the controller will store the value of ACCEPT_PTR in FIFO B as

END_ADD (CS8 in Figure 5). Next, at step 7.30, the controller will clear its match notation. Then, at step 7.31, the controller will set the MB_RD_ADD and BYTE_POS_IN to the values stored in ACCEPT_PTR and proceed to step 7.20 so the byte immediately following the byte identified by ACCEPT_PTR is processed. Once START_ADD is queued in FIFO A and END_ADD is queued in FIFO B, the controller will be able to appropriately modify outgoing packets because it will know the boundaries of the longest matching string in the packet to be modified.

From step 7.20, the controller begins the process of obtaining the next byte. At step 7.20, BYTE_POS_IN is incremented, and then the controller checks whether BYTE_POS_IN is greater than 4 at step 7.21. If BYTE_POS_IN is not greater than 4, then the controller knows that another byte of the current word on an input line of WBC1 needs to be processed. Thus, the controller loops back to step 7.10 to begin processing that byte. If BYTE_POS_IN is greater than 4, then the controller knows that all bytes of the current word have been processed and the next word in MB needs to be obtained. Before getting the next word, the controller checks whether the EOF bit for the current word is high (step 7.22).

If the EOF bit is high, this means that the current word is the last word of the payload, in which case the pattern matching process for the packet is complete. REFSM_ENABLE is unasserted and MB_RD_ADD is set equal to MB_RD_ADD+3 to begin processing the next packet (steps 7.27 and 7.28). Also, to account for the situation where the last byte of the last word of the packet payload is the byte that caused a full match condition to exist, the controller proceeds through steps 7.24, 7.25, and 7.26 that parallel steps 7.18, 7.29, and 7.30. If the EOF bit is not high, this means that the current word is not the last word of the payload and the bytes of the next word need to be processed through the REFSM. Thus, the controller increments MB_RD_ADD (step 7.23) and loops back to step 7.8 to begin processing the word stored in MB at MB_RD_ADD.

The primary task of data path 112 is to output incoming data, and, if necessary, modify that data. The replacement buffer (REPBUF) in the data path stores a replacement string that is to be inserted into the data stream in place of each matching string. Together, the REPBUF and MUX act as a string replacement machine, as will be

explained below. The replacement string stored in REPBUF is provided by a user when the packet processing module is first generated.

The data buffer (DB) will receive the same 35 bits (S1) as does MB. The controller will also use the same DATA_WR_ADD (CS1) to control the writing of words to DB as it does for MB. The DB and the MB will be identical buffers. The controller will use the DB_RD_ADD signal (CS11) to control which words are read from DB.

Word-to-byte converter 2 (WBC2) will operate as WBC1 does; it will break incoming 32 bit words (S7) into 4 bytes and pass those bytes to WBC2's output according to the WBC2_SELECT signal (CS12). Signal S6 will carry the 3 control bits associated with the word read out of DB from address DB_RD_ADD.

A byte is not available for output from the data path until the matching path has already determined whether that byte is part of a matching string. Figure 8 illustrates how this safeguard is achieved. After DB_RD_ADD and BYTE_POS_OUT are initialized (steps 8.1 and 8.2), the controller compares DB_RD_ADD with the MB_RD_ADD stored in BACK_PTR (step 8.3). The controller will not read a word out of DB if the address of that word is greater than or equal to the MB_RD_ADD stored in BACK_PTR. In such cases, the controller waits for the MB_RD_ADD in BACK_PTR to increase beyond DB_RD_ADD. When DB_RD_ADD is less than MB_RD_ADD in BACK_PTR, the controller proceeds to step 8.4 and checks whether the matching path has found a match (is FIFOA empty?). If a match has not been found by the matching path, the controller follows steps 8.6 through 8.11 to output the bytes of that word.

At step 8.6, DB_RD_ADD is asserted, thereby passing the word stored in DB at that address to WBC2 (S7). At step 8.7, WBC2_SELECT is set equal to BYTE_POS_OUT to thereby cause the byte identified by BYTE_POS_OUT to be passed to the WBC2 output (S9). Thereafter, at step 8.8, MUX_SELECT is asserted to pass the output of WBC2 to the output of the MUX (S10). Then, the controller increments BYTE_POS_OUT and repeats steps 8.7 through 8.10 until each byte of the current word is passed through the MUX. When all bytes have been passed through the MUX, DB_RD_ADD is incremented (step 8.11) and the controller loops back to step 8.2.

If step 8.4 results in a determination that there is a START_ADD queued in FIFOA, then the controller compares DB_RD_ADD with the MB_RD_ADD stored with the START_ADD at the head of FIFOA

(step 8.5). IF DB_RD_ADD is less than the MB_RD_ADD stored with START_ADD, steps 8.6 through 8.11 are followed because the word at DB_RD_ADD is not part of a matching string. However, if DB_RD_ADD equals the MB_RD_ADD stored with the dequeued START_ADD, then the controller next needs to identify which byte of the current word (the word at DB_RD_ADD) is the starting byte of the matching string. Thus, at step 8.13 (after START_ADD is dequeued from FIFOA at step 8.12), the controller compares BYTE_POS_OUT with the BYTE_POS_IN stored in START_ADD. IF BYTE_POS_OUT does not equal the BYTE_POS_IN stored in START_ADD, then that byte is not part of the matching string and the controller follows steps 8.14 through 8.16 to pass that byte to the MUX output. Steps 8.14 through 8.16 parallel steps 8.7 through 8.9. Eventually, when the controller returns to step 8.13, BYTE_POS_OUT will match the BYTE_POS_IN stored with the dequeued START_ADD. When this occurs, the controller initiates the string replacement process at step 8.17.

At step 8.17, the controller asserts REPBUF_ENABLE (CS13), and then asserts MUX_SELECT such that the output (S8) of replacement buffer (REPBUF) is passed to the MUX output. When REPBUF is enabled, it begins outputting bytes of the replacement string stored therein. Because MUX_SELECT is asserted to pass S8 to the MUX output, the data path will insert the replacement string stored in REPBUF in the data path. By passing the replacement string to the MUX output rather than the matching string, the data path thereby replaces the matching string in the data stream with the replacement string. Figure 9 illustrates the operation of REPBUF.

REPBUF will have an array that stores the bytes of the replacement string. The pointer ARRAY_RD_ADD will identify which byte of the replacement string is to be outputted. After ARRAY_WR_ADD is initialized at step 9.1, REPBUF checks for the REPBUF_ENABLE signal from the controller (step 9.2). Once REPBUF_ENABLE is received, REPBUF outputs the byte stored at ARRAY_RD_ADD. At step 9.3, REPBUF checks whether ARRAY_RD_ADD points to the last byte of the replacement string. If it does not, ARRAY_RD_ADD is incremented and the next byte is outputted (step 9.6 back to 9.3). When ARRAY_RD_ADD reaches the last byte of the replacement sting, REPBUF_DONE (CS14) is asserted to notify the controller that the entire replacement string has been outputted (step 9.5) and ARRAY_RD_ADD is reset to its initial value.

Returning to Figure 8, after REPBUF_ENABLE is asserted and MUX_SELECT is asserted to pass S8 to S10, the controller waits for the REPBUF_DONE signal from REPBUF (step 8.19). Once REPBUF_DONE is received, the controller determines the next byte to process through the data path. This next byte will be the byte immediately following the last byte of the matching string. The controller achieves this objective by dequeuing END_ADD from the head of FIFOB (step 8.20), setting DB_RD_ADD and BYTE_POS_OUT equal to the values in END_ADD (step 8.21), and returning to step 8.3.

The stream of bytes exiting the MUX (S10) will be ready to exit the SRL once they have been reconverted into a word stream. The byte-to-word converter (BWC) will perform this task. Figure 10 illustrates the operation of BWC. The controller controls the operation of BWC with a BWC_ENABLE signal (CS16). A counter in BWC will track each byte received. The counter is initialized at 0 (step 10.1). BWC will also track how many padding bytes are needed to complete a word. For example, if word being assembled by BWC is to be the last word of the payload and only two bytes are received for that word, two padding bytes will be necessary to complete the word. Thus, the parameter PADDING_COUNT is used as a running representation of how many more bytes are needed by BWC to fill the word. At step 10.2, PADDING_COUNT is set equal to (4-counter). At step 10.3, BWC checks whether the controller has asserted the BWC_ENABLE signal. If it has, BWC receives a byte from MUX output (or possibly a padding byte from the controller via S12) (step 10.4). At step 10.5, BWC checks whether the counter equals 3. When the counter equals 3, BWC will know that the current byte it has received is the last byte of a word. In this situation, the current byte and the other 3 bytes that will have been stored by BWC are passed to the BWC output (S11) as a 32 bit word (step 10.6). Also, because none of the bytes of the word will be padding bytes, PADDING_COUNT will equal 0. BWC provides the PADDING_COUNT signal (CS17) to the controller so that the controller can decide whether a padding byte needs to be passed to BWC via signal S12. From step 10.6, BWC returns to step 1 and resets the counter.

If the counter does not equal 3 at step 10.5, then, at step 10.7, BWC stores the received byte at internal address BWC_ADD where BWC_ADD equals the counter value. Thereafter, the counter is incremented (step 10.8) and BWC returns to step 2.

Figure 11 illustrates how the controller processes the PADDING_COUNT signal from BWC to determine whether a padding byte needs to be provided to BWC. At step 11.1, the controller receives PADDING_COUNT from BWC, and at step 11.2, the controller sets the parameter TRACK_PADDING equal to PADDING_COUNT. Thereafter, the controller checks whether the word being built by BWC is to be the last word of the packet's payload (step 11.3). Because of the replacement process, the byte length of the payload may be altered, which may result in the need to use padding bytes to fill the last word of the payload. If the word being built is to be the last word of the payload, then at step 11.4, the controller checks whether TRACK_PADDING is greater than 0. If it is, a padding byte is sent to BWC (S12) at step 11.5, TRACK_PADDING is decremented (step 11.6), and the controller returns to step 11.4. If 11.4 results in a determination that TRACK_PADDING equals 0, then no padding bytes are needed and the controller returns to step 11.1.

Also, the string replacement process may result in the need to alter the headers and trailers for a packet. The controller is configured to make the necessary changes to the headers and trailers. The words exiting BWC via S11 will be passed to the protocol wrapper 102 for eventual output. Control bits for the outgoing words are asserted by the controller as signal S13 and passed to the protocol wrapper 102.

Now that the packet processing module has been described, attention can be turned toward the hardware within which it is implemented. A preferred platform for the packet processor is Washington University's Field-Programmable Port Extender (FPX). However, it must be noted that the present invention can be implemented on alternate platforms, provided that the platform includes a PLD with supporting devices capable of reprogramming the PLD with different modules.

Details about the FPX platform are known in the art. See, for example, Lockwood, John et al., "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)", ACM International Symposium on Field Programmable Gate Arrays (FPGA 2001), Monterey, CA, February 11-12, 2001; See also, Lockwood, John, "Evolvable Internet Hardware Platforms", NASA/DoD Workshop on Evolvable Hardware (EHW'01), Long Beach, CA, July 12-14, 2001, pp. 271-279; and Lockwood, John et al., "Field Programmable Port Extender

(FPX) for Distributed Routing & Queuing", ACM International Symposium of Field Programmable Gate Arrays (FPGA 2000), Monterey, CA, February 2000, pp. 137-144, the disclosures of which are hereby incorporated by reference. A diagram of the FPX is shown in Figure 12. The main components of the FPX 120 are the Reprogrammable Application Device (RAD) 122 and the Network Interface Device (NID) 124.

The RAD 122 is a field programmable gate array (FPGA). A preferred FPGA is the Xilinx XCV 100E manufactured by the Xilinx Corp. of San Jose, CA. However, any FPGA having enough gates thereon to handle the packet processing module of the present invention would be suitable. Programmed into the RAD 122 will be a packet processing module as described in connection with Figures 3-11. In a preferred embodiment, the RAD 122 can be programmed with two modules, one to handle incoming traffic (data going from the line card to the switch) and one to handle outgoing traffic (data going from the switch back out to the line card). For ingress and egress processing, one set of SRAM and SDRAM is used to buffer data as it arrives, while the other SRAM and SDRAM buffers data as it leaves. However, it should be noted that the RAD 122 can be implemented with any number of modules depending upon the number of gates on the FPGA.

The NID 124 interfaces the RAD with the outside world by recognizing and routing incoming traffic (which may be either coming from the switch or the line card) to the appropriate module and recognizing and routing outgoing traffic (which may be either going to the switch or the line card) to the appropriate output. The NID is also preferably an FPGA but this need not be the case. Another task of the NID 124 is to control the programming of the RAD. When the reconfiguration device transmits configuration information to the packet processor to reprogram the packet scanner with a new module, the NID 124 will recognize the configuration information as configuration information by reading the header that the reconfiguration device includes in the packets within which the configuration information resides. As the NID receives configuration information, the configuration information will be stored in the RAD programming SRAM 126. Once the NID has stored all of the configuration information in the RAD Programming SRAM, the NID will wait for an instruction packet from the reconfiguration device that instructs the NID to reprogram the RAD with the module defined by the configuration information stored in the SRAM 126. Once in receipt of

the instruction packet, the NID loads the configuration information into the RAD by reading the configuration information out of the SRAM 126 and writing it to the reconfiguration ports of the FPGA.

Another feature of the FPX that makes it desirable for use with the present invention is that the FPX is capable of partially reprogramming the RAD while the RAD is still capable of carrying out tasks with the existing module. The FPX supports partial reprogramming of the RAD by allowing configuration streams to contain commands that specify only a portion of the logic on the RAD is to be programmed. Rather than issue a command to reinitialize the device, the NID just writes frame of configuration information to the RAD's reprogramming port. As such, the existing module on the RAD can continue processing packets during the partial configuration.

An overview of the reconfiguration device of the present invention is shown in Figure 13. Main components of the reconfiguration device are a compiler which receives input from a user and generates the configuration information therefrom that is used to reprogram the packet processor, and a transmitter which communicates the configuration information to the packet processor over the network. The reconfiguration device is preferably implemented on a general purpose computer connected to the network, wherein the compiler is preferably software resident thereon, and wherein the transmitter utilizes the network interface also resident thereon. However, alternative implementations would be readily recognizable by those of ordinary skill in the art.

The compiler of the present invention is a powerful tool that allows users to reprogram the reprogrammable packet processor with minimum effort. All that a user has to do is provide the compiler with a data pattern and an action command, and the compiler automates the intensive tasks of designing the module and creating the configuration information necessary to program that module into the packet processor. This streamlined process provides flexibility in reprogramming high-speed packet scanners that was previously unknown in the art.

As an input, the compiler receives two items from a user: (1) the regular expression that defines the data pattern against which packets will be scanned, and (2) the action command which specifies how the packet processor is to respond when packets having a matching string are found. From this input information, the

compiler generates the two dynamic components of Figure 5 - the pattern matching state machine (REFSM) and the replacement buffer (REPBUF). The REFSM will be tuned to determine whether data applied thereto includes a string that matches the user-specified data pattern, and, when the action command specifies a string replacement operation, the REPBUF will be tuned to output a replacement string in accordance with the user-specified string replacement command when activated by the controller.

Also, the compiler will retrieve VHDL representations of the static components of Figures 3-5 that are stored in memory (the protocol wrapper, the twin word buffers MB and DB, the word-to-byte converters WBC1 and WBC2, the controller, the MUX, and the byte-to-word converter BWC). The compiler will integrate the dynamically-created components with the static components to create a logical representation (preferably a VHDL representation) of the packet processing module. FPGA synthesis tools available in the art can convert the VHDL representation of the module into a bitmap operable to program a FPGA with the module. The bitmap of the module serves as the configuration information to be transmitted over the network to the packet processor.

The transmitter operates to packetize the configuration information so it can be communicated over the network to the packet processor. Packetization of data destined for a computer network is well-known in the art and need not be repeated here. However, it should be noted that the transmitter needs to include information in the headers of the packets containing configuration information that will allow the packet processor to recognize those packets as containing configuration information (so that the packet processor can then reprogram itself with that configuration information).

Figure 14 illustrates the operation of the compiler of the present invention. At step 14.1, the compiler receives N lines of input from a user. This input may come either directly from a user via an input device such as a keyboard, it may come indirectly from a user via a web interface, or it may come indirectly from a user via additional software. Each line k of input may specify a different data pattern and action command. Preferably, this input is provided in RE and awk format. Included in Appendix A is an example of input that a user can provide to the compiler. The example shown in Appendix A is a search and replace operation wherein the data pattern

(defined by the RE) is "t.*t" and the replacement string is "this is a test". The compiler will generate configuration information from this input that defines a module operable to detect a string in a packet that matches the pattern "t.*t" and then replace that string with "this is a test".

A high level script called BuildApp is run by the compiler to begin the generation of the configuration information. The code for BuildApp is also included in Appendix A. Steps 14.2 through 14.10 are performed by BuildApp. After index k is initialized to 1 at step 14.2, the compiler sets out to generate the pattern matching state machine (REFSM) and the string replacement machine (REPBUF).

An important tool used by the present invention in the automated creation of the REFSM is the lexical analyzer generator. A lexical analyzer generator is a powerful tool that is executable to receive a regular expression and generate a logical representation of pattern matching state machine therefrom that is operable to determine whether an input stream includes a string that matches the data pattern defined by the regular expression. Lexical analyzer generators are known in the art, and the inventors herein have found that the lexical analyzer generator known as JLex is an excellent lexical analyzer generator for use in connection with the present invention. JLex is publicly-available software developed by Elliot Joel Berk that can be obtained over the Internet from the website http://www.cs.princeton.edu/~appel/modern/java/JLex/.

At step 14.3, the compiler converts line k of the user input into a format readable by the lexical analyzer generator for creating the logical representation of the pattern matching state machine. Preferably, when JLex is used as the lexical analyzer generator, step 14.3 operates to convert line k of the input into the format used by JLex. A script called CreateRegEx is called by BuildApp to perform this task. The code for CreateRegEx is included in Appendix A. Appendix A also includes the output of CreateRegEx for the above example where the RE input is "t.*t".

At step 14.4, the lexical analyzer generator is executed to create a representation of the pattern matching state machine (REFSM) that is tuned with the data pattern defined by the regular expression found in line k of the user input. If JLex is used as the lexical analyzer generator, JLex will create a Java representation of REFSM.

Appendix A further includes the Java representation of the pattern matching state machine for the exemplary RE of "t.*t" (jlex_in.java).

Thus, at step 14.5 an additional operation is needed to convert the Java representation of the REFSM to a VHDL representation of the pattern matching state machine. A script called StateGen will parse the Jlex output (jlex_in.java) to create the VHDL representation of the pattern matching state machine. StateGen is also included in Appendix A. The VHDL entity created by StateGen is saved as RegEx_FSM{k}.vhd (wherein k is the line of user input from which the REFSM was generated). Appendix A also includes the VHDL code for the pattern matching state machine made from the example where the RE is "t.*t" (RegEx_FSM1.vhd).

At step 14.6, the compiler generates a VHDL representation of the replacement buffer (REPBUF) from line k of the user input. A script called ReplaceBufGen (see Appendix A) will control the creation of the replacement buffer. The VHDL representation of the replacement buffer will operate as described in connection with Figure 8. Appendix A also includes the VHDL code for the REPBUF in the above example where the replacement string is "this is a test".

After the dynamic components for line k=1 of the user input are created, at step 14.7, the compiler checks whether there is another line of input. If there is, the compiler proceeds to step 14.8 to increment k, and then loops back to steps 14.3 through 14.6. Once dynamic components have been generated for all lines N of user input, the compiler will have VHDL representations of all N REFSMs and N REPBUFs.

Next, at step 14.9, the compiler, through the BuildApp script, defines the interconnections and signals that will be passed between all of the static and dynamic components of the search and replace logic (SRL). VHDL representations of the static components of the SRL will be stored in memory accessible by the compiler. Appendix A includes the VHDL code for these static components (controller.vhd (which encompasses the controller, word-to-byte converters, and MUX), character.buf.vhd (which encompasses the word buffers), and wrd_bldr.vhd (which defines the byte-to-word converter)). The VHDL representation of the SRL submodule is listed in Appendix A as RegEx_App.vhd. Further, the compiler, through the BuildApp script, defines the interconnections and signals that will be passed between the various wrappers and the SRL to create a VHDL representation of

the packet scanning module. VHDL code for the wrappers will also be stored in memory accessible by the compiler. Appendix A includes the VHDL code for the wrappers and Appendix A includes the resultant VHDL code for the packet scanning module (regex_module.vhd).

Then, at step 14.10, the compiler through BuildApp creates a project file which includes a list of the file names for the VHDL representations of all dynamic and static components of the module. BuildApp calls a script named MakeProject to carry out this task. MakeProject is included in Appendix A, as is its output file RegEx_App.prj.

Further, at step 14.11, the compiler will synthesize the components listed in the project file to create a backend representation of the module. Synthesis tools known in the art may be used for this task, and the inventors have found that the synthesis tool Synplicity Synplify Pro from Synplicity, Inc. of Sunnyvale, CA, is highly suitable. Synplicity Synplify Pro is available on-line at http://www.synplicity.com. The backend module representation created by the synthesis tool is then provided to a backend conversion tool for the reprogrammable hardware (preferably a FPGA conversion tool such as a Xilinx backend conversion tool) to generate a bitmap that is operable to program the packet scanning module into the reprogrammable hardware. This bitmap is the configuration information that defines the module programmed into the PLD of the packet scanner, and may subsequently be transmitted over the network to the packet scanner.

While the present invention has been described above in relation to its preferred embodiment, various modifications may be made thereto that still fall within the invention's scope, as would be recognized by those of ordinary skill in the art.

For example, the packet processing system of the present invention has been described wherein its environment is a computer network and the data stream it processes is a stream of data packets transmitted over the network. However, this need not be the case. The packet processing system of the present invention may be used to process any data stream, no matter its source. For example, the present invention can be used to process streaming data being read from a data source such as a disk drive, a tape drive, a packet radio, a satellite receiver, a fiber optic cable, or other such media.

Also, the SRL used by the packet processing module has been described wherein a single REFSM is used to scan payload bytes. To speed the operation of the SRL, a plurality of REFSMs, each keyed with the same data pattern, may be implemented in parallel. Figure 13 illustrates how the matching path 110 of the SRL can implement parallel REFSMs. Each REFSM can be used to process the byte stream starting from a different byte. For a byte stream $\{B_1, B_2, . . . B_N . . .B_M\}$, the controller can activate the MB_RD_ADD(1) and WBC1_SELECT)(1) such that the byte stream $\{B_1 . . . B_M\}$ is provided to REFSM(1), activate MB_RD_ADD(2) and WBC1_SELECT(2) such that the byte stream $\{B_2 . . . B_M\}$ is passed to REFSM(2), activate MB_RD_ADD(3)and WBC1_SELECT(3) such that the byte stream $(B_3 . . . B_M\}$ is passed to REFSM(3), and so on for N REFSMs. In this configuration, time will not be wasted processing a non-matching string starting at byte 1 because another REFSM will already be processing a potentially matching string starting at byte 2. The controller can be modified to account for situations where more than one REFSM detects a match. For example, where REFSM(1) has found a match for string $\{B_1, B_2, . . . B_6\}$ and REFSM(2) has found a match for string $(B_2, B_3, . . . B_6\}$, the controller can be designed to choose the longest matching string (i.e., $\{B_1, B_2, . . . , B_6\}$.

Also, each parallel REFSM in Figure 15 can be keyed with a different data pattern. The same byte stream can be provided to each REFSM, and the controller can process each REFSM_STATUS signal to determine which data patterns are present in the data stream.

Further, the packet processor has been described above wherein the action performed thereby when a match is found is a straight replacement operation. However, a wide variety of additional actions may also be readily implemented. Rather than replace a matching string, the processor can be configured to drop a packet that includes a matching string by not outputting such a packet from the processing module. Also, the data path of the processing logic can be configured to output a notification packet addressed to an interested party when a matching string is found. Such a notification packet may include a copy of the packet that includes the matching string. Also, because the present invention allows the packet processor to not only identify that a match has occurred but also identify the matching string as well as the data pattern with which a string is matched, such information can be used to gather

statistics about the data stream. Appropriate signals can be passed to a statistic-keeping device that monitors the content of the data stream.

Further still, when a back substitution operation is desired rather than a straight substitution operation (in a back substitution operation, the replacement string will include at least one copy of the matching string), the algorithm of Figure 8 can be modified in accordance with Figure 16. Figure 16 picks up from step 8.17 in Figure 8. In back substitution, the replacement string will include a byte that indicates the matching string is to be inserted into the data stream. When this byte is outputted from REPBUF, the controller freezes REPBUF and reads and passes the matching string from DB to WBC2 to MUX output. Once the matching string is inserted in the data stream, the controller reactivates REPBUF to continue the outputting of the replacement string.

Also, the matching path of SRL 104 can be configured for case insensitivity (wherein upper case letters are treated the same as lower case letters) by adding a case converter between the output of WBC1 and the input of REFSM. The case converter will be operable to convert each incoming byte to a common case (either all caps or all lower case) that matches the case of the data pattern with which the REFSM is tuned. For example, the case converter would convert the stream "abcDefghIJKlm" to stream "ABCDEFGHIJKLM" when case insensitivity is desired and the REFSM is tuned with a data pattern defined by all capital letters.

Further, the packet processor has been described wherein it is implemented as a stand-alone device on the FPX that interfaces a line card and a packet switch. However, one of ordinary skill in the art would readily recognize that the reprogrammable packet processor may be implemented as an internal component of any network processing device (such as a packet switch).

Further still, the packet processor of the present invention may be used with all manner of networks, such as the Internet or various local area networks (LANs) including wireless LANs. For example, the packet processor can be fitted with a wireless transceiver to receive and transmit wireless data to thereby integrate the packet processor with a wireless network (wireless transceivers being known in the art).

These and other modifications to the preferred embodiment will be recognizable upon review of the teachings herein. As such, the full scope of the present invention is to be defined by the appended claims in view of the description above, attached figures, and Appendix.

APPENDIX A

***Character Set**
***SNRList**
***BuildApp**
***jlex_in**
***jlex_in.java**
***CreateRegEx**
***StateGen**
***ReplaceBufGen**
***regex_app.vhd**
***replace_buf1.vhd**
***controller.vhd**
***wrd_bldr.vhd**
***character_buf.vhd**
***regex_fsm1.vhd**
***rad_loopback_core.vhd**
***rad_loopback.vhd**
***loopback_module.vhd**
***blink.vhd**
***regex_module.vhd**
***MakeProject**
***regex_app.prj**

Character Set

```
0:NUL:"00000000"
1:SOH:"00000001"
2:STX:"00000010"
3:ETX:"00000011"
4:EOT:"00000100"
5:ENQ:"00000101"
6:ACK:"00000110"
7:BEL:"00000111"
8:BS:"00001000"
9:HT:"00001001"
10:LF:"00001010"
11:VT:"00001011"
12:FF:"00001100"
13:CR:"00001101"
14:SO:"00001110"
15:SI:"00001111"
16:DLE:"00010000"
17:DC1:"00010001"
18:DC2:"00010010"
19:DC3:"00010011"
20:DC4:"00010100"
21:NAK:"00010101"
22:SYN:"00010110"
23:ETB:"00010111"
24:CAN:"00011000"
25:EM:"00011001"
26:SUB:"00011010"
27:ESC:"00011011"
28:FSP:"00011100"
29:GSP:"00011101"
30:RSP:"00011110"
31:USP:"00011111"
32:' ':"00100000"
33:'!':"00100001"
34:'"':"00100010"
35:'#':"00100011"
36:'$':"00100100"
37:'%':"00100101"
38:'&':"00100110"
39:''':"00100111"
40:'(':"00101000"
41:')':"00101001"
42:'*':"00101010"
43:'+':"00101011"
44:',':"00101100"
45:'-':"00101101"
46:'.':"00101110"
47:'/':"00101111"
48:'0':"00110000"
49:'1':"00110001"
50:'2':"00110010"
51:'3':"00110011"
52:'4':"00110100"
53:'5':"00110101"
54:'6':"00110110"
55:'7':"00110111"
56:'8':"00111000"
57:'9':"00111001"
58:':':"00111010"
59:';':"00111011"
60:'<':"00111100"
```

```
61:'=':"00111101"
62:'>':"00111110"
63:'?':"00111111"
64:'@':"01000000"
65:'A':"01000001"
66:'B':"01000010"
67:'C':"01000011"
68:'D':"01000100"
69:'E':"01000101"
70:'F':"01000110"
71:'G':"01000111"
72:'H':"01001000"
73:'I':"01001001"
74:'J':"01001010"
75:'K':"01001011"
76:'L':"01001100"
77:'M':"01001101"
78:'N':"01001110"
79:'O':"01001111"
80:'P':"01010000"
81:'Q':"01010001"
82:'R':"01010010"
83:'S':"01010011"
84:'T':"01010100"
85:'U':"01010101"
86:'V':"01010110"
87:'W':"01010111"
88:'X':"01011000"
89:'Y':"01011001"
90:'Z':"01011010"
91:'[':"01011011"
92:'\':"01011100"
93:']':"01011101"
94:'^':"01011110"
95:'_':"01011111"
96:'`':"01100000"
97:'a':"01100001"
98:'b':"01100010"
99:'c':"01100011"
100:'d':"01100100"
101:'e':"01100101"
102:'f':"01100110"
103:'g':"01100111"
104:'h':"01101000"
105:'i':"01101001"
106:'j':"01101010"
107:'k':"01101011"
108:'l':"01101100"
109:'m':"01101101"
110:'n':"01101110"
111:'o':"01101111"
112:'p':"01110000"
113:'q':"01110001"
114:'r':"01110010"
115:'s':"01110011"
116:'t':"01110100"
117:'u':"01110101"
118:'v':"01110110"
119:'w':"01110111"
120:'x':"01111000"
121:'y':"01111001"
```

```
122:'z':"01111010"
123:'{':"01111011"
124:'|':"01111100"
125:'}':"01111101"
126:'~':"01111110"
127:DEL:"01111111"
```

SNRList

```
s/t.*t/this is a test/g
```

BuildApp

```
#!/bin/gawk -f

BEGIN {

  outfile = "regex_app.vhd"
  infile  = ARGV[1]

  print "library ieee;" > outfile
  print "use ieee.std_logic_1164.all;" > outfile
  print "" > outfile
  print "entity regex_app is" > outfile
  print "  port(clk              : in  std_logic;" > outfile
  print "       reset_l          : in  std_logic;" > outfile
  print "       enable_l         : in  std_logic;" > outfile
  print "       ready_l          : out std_logic;" > outfile
  print "" > outfile
  print "       dataEn_out_appl : in  std_logic;" > outfile
  print "       d_out_appl       : in  std_logic_vector(31 downto 0);" >
outfile
  print "       sof_out_appl     : in  std_logic;" > outfile
  print "       eof_out_appl     : in  std_logic;" > outfile
  print "       sod_out_appl     : in  std_logic;" > outfile
  print "       tca_appl_in      : in  std_logic;" > outfile
  print "" > outfile
  print "       dataEn_appl_in   : out std_logic;" > outfile
  print "       d_appl_in        : out std_logic_vector(31 downto 0);" >
outfile
  print "       sof_appl_in      : out std_logic;" > outfile
  print "       eof_appl_in      : out std_logic;" > outfile
  print "       sod_appl_in      : out std_logic;" > outfile
  print "       tca_out_appl     : out std_logic);" > outfile
  print "end regex_app;" > outfile
  print "" > outfile
  print "" > outfile
  print "architecture regex_app_arch of regex_app is" > outfile
  print "" > outfile

  #print the flopped signals
  print "  signal dataEn_in              : std_logic;" > outfile
  print "  signal data_in                : std_logic_vector(31 downto
0);" > outfile
  print "  signal sof_in                 : std_logic;" > outfile
  print "  signal eof_in                 : std_logic;" > outfile
  print "  signal sod_in                 : std_logic;" > outfile
  print "  signal tca_in                 : std_logic;" > outfile
  print "  signal dataEn_out             : std_logic;" > outfile
  print "  signal data_out               : std_logic_vector(31 downto
0);" > outfile
  print "  signal sof_out                : std_logic;" > outfile
  print "  signal eof_out                : std_logic;" > outfile
  print "  signal sod_out                : std_logic;" > outfile
  print "  signal tca_out                : std_logic;" > outfile
  print " " > outfile

# create the list of signals needed to daisy chain all the modules
  n = 0
  while (getline line < infile)
  {
    n++
    print "-- signals for machine #" n > outfile
    print "  signal ready_l" n "          : std_logic;" > outfile
```

```
    print "" > outfile
    print "  signal regex_en" n "         : std_logic;" > outfile
    print "  signal regex_in" n "         : std_logic_vector(7 downto
0);" > outfile
    print "  signal running" n "          : std_logic;" > outfile
    print "  signal accepting" n "        : std_logic;" > outfile
    print "  signal reseting" n "         : std_logic;" > outfile
    print "" > outfile
    print "  signal start_replacing" n " : std_logic;" > outfile
    print "  signal done_replacing" n "  : std_logic;" > outfile
    print "  signal new_char" n "        : std_logic_vector(7 downto
0);" > outfile
    print "" > outfile

    if (n != 1)
    {
      print "  signal tca_out_appl" n "     : std_logic;" > outfile
      print "  signal dataEn_out_appl" n "  : std_logic;" > outfile
      print "  signal d_out_appl" n "       : std_logic_vector(31 downto
0);" > outfile
      print "  signal sof_out_appl" n "     : std_logic;" > outfile
      print "  signal eof_out_appl" n "     : std_logic;" > outfile
      print "  signal sod_out_appl" n "     : std_logic;" > outfile
      print "" > outfile
    }

  }
  totalMachines = n

  close(infile)
  print "" > outfile
  print "" > outfile
  print "" > outfile


# create instantiations for each component
  n=0
  while (getline line < infile)
  {
    n++
    len = split(line, snr, "/")

    if (len != 4)
    {
      print "\nERROR (line "n"): incorrect SNRlist file format!!\n"
      print "  FORMAT:"
      print "     '{regular expression}'  '{replacement string}'"
      print ""
      break
    }

    # create the necessary files for the job
    system("./createRegex -v regex='" snr[2] "' -v n='"n"'")
    system("./replaceBufGen -v replacement='" snr[3] "' -v n='"n"'")


    print "  component regex_fsm" n > outfile
    print "    port(clk              : in  std_logic;" > outfile
    print "          reset_l         : in  std_logic;" > outfile
    print "          regex_en        : in  std_logic;" > outfile
```

```
    print "         regex_in       : in  std_logic_vector(7 downto
0);" > outfile
    print "         running        : out std_logic;" > outfile
    print "         accepting      : out std_logic;" > outfile
    print "         reseting       : out std_logic);" > outfile
    print "  end component;" > outfile
    print "" > outfile
    print "" > outfile
    print "" > outfile
    print "  component replace_buf" n > outfile
    print "    port(clk             : in  std_logic;" > outfile
    print "         reset_l         : in  std_logic;" > outfile
    print "         start_replacing : in  std_logic;" > outfile
    print "         done_replacing  : out std_logic;" > outfile
    print "         new_char        : out std_logic_vector(7 downto
0));" > outfile
    print "  end component;" > outfile
    print "" > outfile
    print "" > outfile
    print "" > outfile
  }
  close(infile)

  print "  component controller" > outfile
  print "    port(clk             : in  std_logic;" > outfile
  print "         reset_l         : in  std_logic;" > outfile
  print "         enable_l        : in  std_logic;" > outfile
  print "         ready_l         : out std_logic;" > outfile
  print "" > outfile
  print "         dataEn_out_appl : in  std_logic;" > outfile
  print "         d_out_appl      : in  std_logic_vector(31 downto
0);" > outfile
  print "         sof_out_appl    : in  std_logic;" > outfile
  print "         eof_out_appl    : in  std_logic;" > outfile
  print "         sod_out_appl    : in  std_logic;" > outfile
  print "         tca_appl_in     : in  std_logic;" > outfile
  print "" > outfile
  print "         dataEn_appl_in  : out std_logic;" > outfile
  print "         d_appl_in       : out std_logic_vector(31 downto
0);" > outfile
  print "         sof_appl_in     : out std_logic;" > outfile
  print "         eof_appl_in     : out std_logic;" > outfile
  print "         sod_appl_in     : out std_logic;" > outfile
  print "         tca_out_appl    : out std_logic;" > outfile
  print "" > outfile
  print "         regex_en        : out std_logic;" > outfile
  print "         regex_in        : out std_logic_vector(7 downto 0);" >
outfile
  print "         running         : in  std_logic;" > outfile
  print "         accepting       : in  std_logic;" > outfile
  print "         reseting        : in  std_logic;" > outfile
  print "" > outfile
  print "         start_replacing : out std_logic;" > outfile
  print "         done_replacing  : in  std_logic;" > outfile
  print "         new_char        : in  std_logic_vector(7 downto
0));" > outfile
  print "  end component;" > outfile
  print "" > outfile
  print "" > outfile
  print "" > outfile
```

```
  print "begin" > outfile
  print "" > outfile


  for(n=1; n<=totalMachines; n++)
  {
    print "  regular_expression_machine" n " : regex_fsm" n > outfile
    print "  port map(clk              => clk," > outfile
    print "            reset_l          => reset_l," > outfile
    print "            regex_en         => regex_en" n "," > outfile
    print "            regex_in         => regex_in" n "," > outfile
    print "            running          => running" n "," > outfile
    print "            accepting        => accepting" n "," > outfile
    print "            reseting         => reseting" n ");" > outfile
    print "" > outfile
    print "" > outfile

    print "  replacement_buffer" n " : replace_buf" n > outfile
    print "  port map(clk              => clk," > outfile
    print "            reset_l          => reset_l," > outfile
    print "            start_replacing => start_replacing" n "," >
outfile
    print "            done_replacing   => done_replacing" n "," > outfile
    print "            new_char         => new_char" n ");" > outfile
    print "" > outfile
    print "" > outfile

    print "  controller" n " : controller" > outfile
    print "  port map(clk              => clk," > outfile
    print "            reset_l          => reset_l," > outfile
    print "            enable_l         => enable_l," > outfile
    print "            ready_l          => ready_l" n "," > outfile
    print "" > outfile


#################$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

#    if (n==1) m=""
#      else m=n

    if (n==1)
    {
      print "            dataEn_out_appl => dataEn_in,"  > outfile
      print "            d_out_appl      => data_in,"    > outfile
      print "            sof_out_appl    => sof_in,"     > outfile
      print "            eof_out_appl    => eof_in,"     > outfile
      print "            sod_out_appl    => sod_in,"     > outfile
    }
    else
    {
      print "            dataEn_out_appl => dataEn_out_appl" n "," >
outfile
      print "            d_out_appl      => d_out_appl" n "," > outfile
      print "            sof_out_appl    => sof_out_appl" n "," > outfile
      print "            eof_out_appl    => eof_out_appl" n "," > outfile
      print "            sod_out_appl    => sod_out_appl" n "," > outfile
    }

    if (n==totalMachines)
      print "            tca_appl_in     => tca_in,       -- take in from
```

```
downstream mod" > outfile
    else
      print "             tca_appl_in     => tca_out_appl" n+1 ",      --
take in from downstream mod" > outfile

    print "" > outfile



    if (n==totalMachines)
    {
      print "             dataEn_appl_in  => dataEn_out,    -- send to
downstream mod" > outfile
      print "             d_appl_in       => data_out,"    > outfile
      print "             sof_appl_in     => sof_out," > outfile
      print "             eof_appl_in     => eof_out," > outfile
      print "             sod_appl_in     => sod_out," > outfile
    }
    else
    {
      print "             dataEn_appl_in  => dataEn_out_appl" n+1 "," >
outfile
      print "             d_appl_in       => d_out_appl" n+1 ","       >
outfile
      print "             sof_appl_in     => sof_out_appl" n+1 ","     >
outfile
      print "             eof_appl_in     => eof_out_appl" n+1 ","     >
outfile
      print "             sod_appl_in     => sod_out_appl" n+1 ","     >
outfile
    }

    if (n==1)
      print "             tca_out_appl    => tca_out," > outfile
    else
      print "             tca_out_appl    => tca_out_appl" n "," > outfile
    print "" > outfile



##################$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


    print "-- SIGNALS FOR SEARCH AND REPLACE" > outfile
    print "            regex_en         => regex_en" n "," > outfile
    print "            regex_in         => regex_in" n "," > outfile
    print "            running          => running" n "," > outfile
    print "            accepting        => accepting" n "," > outfile
    print "            reseting         => reseting" n "," > outfile
    print "" > outfile
    print "            start_replacing => start_replacing" n "," >
outfile
    print "            done_replacing   => done_replacing" n "," > outfile
    print "            new_char         => new_char" n ");" > outfile
    print "" > outfile
    print "" > outfile

    print
"------------------------------------------------------------------------
------" > outfile
```

```
    print "" > outfile
  }

    print "  flop_signals: process (clk)" > outfile
    print "  begin" > outfile
    print "    if clk'event and clk = '1' then" > outfile
    print "      dataEn_in      <= dataEn_out_appl;" > outfile
    print "      data_in        <= d_out_appl;" > outfile
    print "      sof_in         <= sof_out_appl;" > outfile
    print "      eof_in         <= eof_out_appl;" > outfile
    print "      sod_in         <= sod_out_appl;" > outfile
    print "      tca_in         <= tca_appl_in;"  > outfile
    print " " > outfile
    print "      dataEn_appl_in <= dataEn_out;" > outfile
    print "      d_appl_in      <= data_out;"   > outfile
    print "      sof_appl_in    <= sof_out;"    > outfile
    print "      eof_appl_in    <= eof_out;"    > outfile
    print "      sod_appl_in    <= sod_out;"    > outfile
    print "      tca_out_appl   <= tca_out;"    > outfile
    print "    end if;" > outfile
    print "  end process flop_signals;" > outfile


    print "" > outfile
    print "" > outfile


## make sure ALL machines are ready_l
  printf "  ready_l <=" > outfile
  for(n=1; n<totalMachines; n++)
  {
    printf " ready_l" n " and" > outfile
  }
  print " ready_l" n";" > outfile
  print "" > outfile
  print "" > outfile

  print "end regex_app_arch;" > outfile

## now create the project file
  system("./makeProject -v n='" totalMachines "'")


}
```

jlex_in

```
%%
%%
"t".*"t"
{  }
```

jlex_in.java

```
class Yylex {
      private final int YY_BUFFER_SIZE = 512;
      private final int YY_F = -1;
      private final int YY_NO_STATE = -1;
      private final int YY_NOT_ACCEPT = 0;
      private final int YY_START = 1;
      private final int YY_END = 2;
      private final int YY_NO_ANCHOR = 4;
      private final int YY_BOL = 128;
      private final int YY_EOF = 129;
      private java.io.BufferedReader yy_reader;
      private int yy_buffer_index;
      private int yy_buffer_read;
      private int yy_buffer_start;
      private int yy_buffer_end;
      private char yy_buffer[];
      private boolean yy_at_bol;
      private int yy_lexical_state;

      Yylex (java.io.Reader reader) {
            this ();
            if (null == reader) {
                  throw (new Error("Error: Bad input stream
initializer."));
            }
            yy_reader = new java.io.BufferedReader(reader);
      }

      Yylex (java.io.InputStream instream) {
            this ();
            if (null == instream) {
                  throw (new Error("Error: Bad input stream
initializer."));
            }
            yy_reader = new java.io.BufferedReader(new
java.io.InputStreamReader(instream));
      }

      private Yylex () {
            yy_buffer = new char[YY_BUFFER_SIZE];
            yy_buffer_read = 0;
            yy_buffer_index = 0;
            yy_buffer_start = 0;
            yy_buffer_end = 0;
            yy_at_bol = true;
            yy_lexical_state = YYINITIAL;
      }

      private boolean yy_eof_done = false;
      private final int YYINITIAL = 0;
      private final int yy_state_dtrans[] = {
            0
      };
      private void yybegin (int state) {
            yy_lexical_state = state;
      }
      private int yy_advance ()
            throws java.io.IOException {
            int next_read;
```

```
        int i;
        int j;

        if (yy_buffer_index < yy_buffer_read) {
              return yy_buffer[yy_buffer_index++];
        }

        if (0 != yy_buffer_start) {
              i = yy_buffer_start;
              j = 0;
              while (i < yy_buffer_read) {
                    yy_buffer[j] = yy_buffer[i];
                    ++i;
                    ++j;
              }
              yy_buffer_end = yy_buffer_end - yy_buffer_start;
              yy_buffer_start = 0;
              yy_buffer_read = j;
              yy_buffer_index = j;
              next_read = yy_reader.read(yy_buffer,
                          yy_buffer_read,
                          yy_buffer.length - yy_buffer_read);
              if (-1 == next_read) {
                    return YY_EOF;
              }
              yy_buffer_read = yy_buffer_read + next_read;
        }

        while (yy_buffer_index >= yy_buffer_read) {
              if (yy_buffer_index >= yy_buffer.length) {
                    yy_buffer = yy_double(yy_buffer);
              }
              next_read = yy_reader.read(yy_buffer,
                          yy_buffer_read,
                          yy_buffer.length - yy_buffer_read);
              if (-1 == next_read) {
                    return YY_EOF;
              }
              yy_buffer_read = yy_buffer_read + next_read;
        }
        return yy_buffer[yy_buffer_index++];
  }
  private void yy_move_end () {
        if (yy_buffer_end > yy_buffer_start &&
            '\n' == yy_buffer[yy_buffer_end-1])
              yy_buffer_end--;
        if (yy_buffer_end > yy_buffer_start &&
            '\r' == yy_buffer[yy_buffer_end-1])
              yy_buffer_end--;
  }
  private boolean yy_last_was_cr=false;
  private void yy_mark_start () {
        yy_buffer_start = yy_buffer_index;
  }
  private void yy_mark_end () {
        yy_buffer_end = yy_buffer_index;
  }
  private void yy_to_mark () {
        yy_buffer_index = yy_buffer_end;
        yy_at_bol = (yy_buffer_end > yy_buffer_start) &&
                    ('\r' == yy_buffer[yy_buffer_end-1] ||
```

```
                    '\n' == yy_buffer[yy_buffer_end-1] ||
                    2028/*LS*/ == yy_buffer[yy_buffer_end-1] ||
                    2029/*PS*/ == yy_buffer[yy_buffer_end-1]);
    }
    private java.lang.String yytext () {
        return (new java.lang.String(yy_buffer,
            yy_buffer_start,
            yy_buffer_end - yy_buffer_start));
    }
    private int yylength () {
        return yy_buffer_end - yy_buffer_start;
    }
    private char[] yy_double (char buf[]) {
        int i;
        char newbuf[];
        newbuf = new char[2*buf.length];
        for (i = 0; i < buf.length; ++i) {
            newbuf[i] = buf[i];
        }
        return newbuf;
    }
    private final int YY_E_INTERNAL = 0;
    private final int YY_E_MATCH = 1;
    private java.lang.String yy_error_string[] = {
        "Error: Internal error.\n",
        "Error: Unmatched input.\n"
    };
    private void yy_error (int code,boolean fatal) {
        java.lang.System.out.print(yy_error_string[code]);
        java.lang.System.out.flush();
        if (fatal) {
            throw new Error("Fatal Error.\n");
        }
    }
    private int[][] unpackFromString(int size1, int size2, String st)

        int colonIndex = -1;
        String lengthString;
        int sequenceLength = 0;
        int sequenceInteger = 0;

        int commaIndex;
        String workString;

        int res[][] = new int[size1][size2];
        for (int i= 0; i < size1; i++) {
            for (int j= 0; j < size2; j++) {
                if (sequenceLength != 0) {
                    res[i][j] = sequenceInteger;
                    sequenceLength--;
                    continue;
                }
                commaIndex = st.indexOf(',');
                workString = (commaIndex==-1) ? st :
                    st.substring(0, commaIndex);
                st = st.substring(commaIndex+1);
                colonIndex = workString.indexOf(':');
                if (colonIndex == -1) {
                    res[i][j]=Integer.parseInt(workString);
                    continue;
                }
```

```
                    lengthString =
                          workString.substring(colonIndex+1);
                    sequenceLength=Integer.parseInt(lengthString);
                    workString=workString.substring(0,colonIndex);
                    sequenceInteger=Integer.parseInt(workString);
                    res[i][j] = sequenceInteger;
                    sequenceLength--;
              }
        }
        return res;
  }
  private int yy_acpt[] = {
        /* 0 */ YY_NOT_ACCEPT,
        /* 1 */ YY_NO_ANCHOR,
        /* 2 */ YY_NO_ANCHOR,
        /* 3 */ YY_NOT_ACCEPT
  };
  private int yy_cmap[] = unpackFromString(1,130,
"2:10,0,2:2,0,2:102,1,2:11,3:2")[0];

  private int yy_rmap[] = unpackFromString(1,4,
"0,1,2:2")[0];

  private int yy_nxt[][] = unpackFromString(3,4,
"-1,3,-1,1,-1:5,2,3,-1");

  public Yytoken yylex ()
        throws java.io.IOException {
        int yy_lookahead;
        int yy_anchor = YY_NO_ANCHOR;
        int yy_state = yy_state_dtrans[yy_lexical_state];
        int yy_next_state = YY_NO_STATE;
        int yy_last_accept_state = YY_NO_STATE;
        boolean yy_initial = true;
        int yy_this_accept;

        yy_mark_start();
        yy_this_accept = yy_acpt[yy_state];
        if (YY_NOT_ACCEPT != yy_this_accept) {
              yy_last_accept_state = yy_state;
              yy_mark_end();
        }
        while (true) {
              if (yy_initial && yy_at_bol) yy_lookahead = YY_BOL;
              else yy_lookahead = yy_advance();
              yy_next_state = YY_F;
              yy_next_state = yy_nxt[yy_rmap[yy_state]][yy_cmap
 [yy_lookahead]];
              if (YY_EOF == yy_lookahead && true == yy_initial) {
                    return null;
              }
              if (YY_F != yy_next_state) {
                    yy_state = yy_next_state;
                    yy_initial = false;
                    yy_this_accept = yy_acpt[yy_state];
                    if (YY_NOT_ACCEPT != yy_this_accept) {
                          yy_last_accept_state = yy_state;
                          yy_mark_end();
                    }
              }
              else {
```

```
                                if (YY_NO_STATE == yy_last_accept_state) {
                                      throw (new Error("Lexical Error: Unmatched
Input."));
                                }
                                else {
                                      yy_anchor = yy_acpt[yy_last_accept_state];
                                      if (0 != (YY_END & yy_anchor)) {
                                            yy_move_end();
                                      }
                                      yy_to_mark();
                                      switch (yy_last_accept_state) {
                                      case 1:

                                      case -2:
                                            break;
                                      case 2:
                                            {  }
                                      case -3:
                                            break;
                                      default:
                                            yy_error(YY_E_INTERNAL,false);
                                      case -1:
                                      }
                                      yy_initial = true;
                                      yy_state = yy_state_dtrans
 [yy_lexical_state];
                                      yy_next_state = YY_NO_STATE;
                                      yy_last_accept_state = YY_NO_STATE;
                                      yy_mark_start();
                                      yy_this_accept = yy_acpt[yy_state];
                                      if (YY_NOT_ACCEPT != yy_this_accept) {
                                            yy_last_accept_state = yy_state;
                                            yy_mark_end();
                                      }
                                }
                          }
                    }
              }
      }
```

CreateRegEx

```
#!/bin/gawk -f

BEGIN {

  if(regex=="")
  {
    print "\nUSAGE: \n"
    print "  createRegex -v regex={regular expression} [-v n=
{identifier}]\n"
    print "   {regular expression} :"
    print "      - the regular expression you want the machine to look
for\n"
    print ""
    print "   {identifier} :"
    print "      - an optional identifer that will be added to the
entity "
    print "        name of the regular expression machine\n\n\n"
  }

  print "%%" > "jlex_in"
  print "%%" > "jlex_in"

  x = split(regex, chars, "")
  open = 0
  for(i = 1; i <= x; i++)
  {
    if( chars[i] ~ "[A-Za-z0-9 ]")
    {
      if(open == 0)
      {
        open = 1
        printf "\"" chars[i] > "jlex_in"
      }
      else
        printf chars[i] > "jlex_in"
    }
    else
    {
      if(open == 1)
      {
        open = 0
        printf "\"" chars[i] > "jlex_in"
      }
      else
        printf chars[i] > "jlex_in"
    }
  }
  # one last closing bracket
  if(open == 1)
    printf "\"" > "jlex_in"


  print "" > "jlex_in"
  printf "\{ " > "jlex_in"
  printf " \}" > "jlex_in"

  system("java JLex.Main jlex_in > lexOut")
#  system("rm -r jlex_in")

  # check to see if there were any errors created when we tried to
  # create the java file
```

```
  while(getline < "lexOut")
  {
    found = match($0, "rror")
    if( found != 0 )
    {
      print "Error creating regex_fsm" n ".vhd : JLex failed to produce
output"
      print " see file lexOut for more details"
      break
    }
  }
  # only create the vhdl file if there were no errors previously
  if (found == 0)
    system("cat jlex_in.java | ./stateGen -v regex='"regex"' -v
n='"n"'")
#   system("rm -r jlex_in.java")
#   system("rm lexOut")
}
```

StateGen

```
#!/bin/gawk -f
# This script will parse the output of JLex and
# convert the code into a VHDL state machine

BEGIN {

  outfile = "regex_fsm" n ".vhd"

# get to the important part of the input file
############################################
  while(getline)
  {
    found = match($0, "yy_acpt\\[\\]")
    if( found != 0)
    {
      break
    }
  }
  num_states = 0
  while(getline)
  {
    found = match($0, "\\}\\;")
    if( found != 0 )
    {
      # do nothing here... we have found the end of the
      # yy_acpt[] array
      break;
    }
    else
    {
      gsub(/\/\*/, "", $0)
      gsub(/\*\//, "", $0)
      gsub(/\,/, "", $0)
      gsub(/\r/, "", $0)
      split($0, acpt, " ")

      # the yy_acpt contains all the acceptance information
      # each array position on the array represents its
      # respective state.  if a 1 is stored in the array,
      # then it is an accepting state, otherwise it is not.
      if(acpt[2] == "YY_NOT_ACCEPT")
      {
        yy_acpt[acpt[1]] = 0
      }
      else
      {
        yy_acpt[acpt[1]] = 1
      }
      num_states++
    }
  }

###############################################
###############################################


# now get to the cmap array
########################
  while(getline)
  {
    found = match($0, "yy_cmap\\[\\]")
```

```
    if( found != 0 )
    {
      break
    }
  }

# get the line with all the character information
# and put it nicely into the cmap array
##################################################
# if the array spans more than one line in the file
# this will concatenate them all together before
# working with the array
  str = ""
  while(getline)
  {
    found = match($0, "\\)\\[0\\]\\;")
    if( found != 0 )
    {
      str = str $1
      break
    }
    else
    {
      str = str $1
    }
  }
  # clean it up a bit
  gsub(/\"/, "", str)
  gsub(/\)\[0\];/, "", str)
  gsub(/\r/, "", str)

  subs = split(str, tmp, ",")
  cols = 0
  for( i = 1; i <= subs; i++)
  {
    sp = split(tmp[i], tmp2, ":")
    if( sp == 1 )
    {
      yy_cmap[cols] = tmp[i]
      cols++
    }
    else
    {
      for( j = 0; j < tmp2[2]; j++)
      {
        yy_cmap[cols] = tmp2[1]
        cols++
      }
    }
  }

##################################################
##################################################


# get the line with the yy_rmap array
# and break it up nicely
########################################
  while(getline)
  {
    found = match($0, "yy_rmap\\[\\]")
```

```
    if( found != 0 )
    {
      break
    }
  }

  str = ""
  while(getline)
  {
    found = match($0, "\\)\\[0\\]\\;")
    if( found != 0 )
    {
      str = str $1
      break
    }
    else
    {
      str = str $1
    }
  }

  # clean it up a bit
  gsub(/\"/, "", str)
  gsub(/\)\[0\];/, "", str)
  gsub(/\r/, "", str)

  subs = split(str, tmp, ",")
  rows = 0
  for( i = 1; i <= subs; i++)
  {
    sp = split(tmp[i], tmp2, ":")
    if( sp == 1 )
    {
      yy_rmap[rows] = tmp[i]
      rows++
    }
    else
    {
      for( j = 0; j < tmp2[2]; j++)
      {
        yy_rmap[rows] = tmp2[1]
        rows++
      }
    }
  }

#################################################
#################################################


# get the line with the yy_nxt array
# and break it up nicely
#######################################

  while(getline)
  {
    found = match($0, "yy_nxt\\[\\]")
    if( found != 0 )
    {
      break
    }
```

```
}

split($0, tmp, "\\(")
split(tmp[2], tmp2, "\\,")
rows2 =  tmp2[1]
cols2 =  tmp2[2]


str = ""
while(getline)
{
  found = match($0, "\\)\\;")
  if( found != 0 )
  {
    str = str $1
    break
  }
  else
  {
    str = str $1
  }
}

# clean it up a bit
gsub(/\"/, "", str)
gsub(/\)\;/, "", str)

subs = split(str, tmp, ",")
xrow = 0
ycol = 0
for( i = 1; i <= subs; i++)
{
  sp = split(tmp[i], tmp2, ":")
  if( sp == 1 )
  {
    yy_nxt[xrow, ycol] = tmp[i]
    ycol++
    if( ycol == cols2 )
    {
      xrow++
      ycol = 0
    }
  }
  else
  {
    for( j = 0; j < tmp2[2]; j++)
    {
      yy_nxt[xrow, ycol] = tmp2[1]
      ycol++
      if( ycol == cols2 )
      {
        xrow++
        ycol = 0
      }
    }
  }
}

##################################################
```

```
# Now print out the VHDL
################################################

  printf "-- This code was generated on " > outfile
  format = "%a %b %e %H:%M:%S %Z %Y"
  print strftime(format) > outfile
  print "-- by a GAWK script created by James Moscola (4-22-2001)" >
outfile

  print "-- Regular Expression is:    " regex > outfile
  print "" > outfile
  print "library ieee;" > outfile
  print "use ieee.std_logic_1164.all;" > outfile
  print "use ieee.std_logic_arith.all;" > outfile
  print "" > outfile
  print "" > outfile
  print "entity regex_fsm" n " is" > outfile

#######################################
# insert inputs here when determined  #
#######################################
  print "  port(clk        : in  std_logic;" > outfile
  print "       reset_l    : in  std_logic;" > outfile
  print "       regex_en   : in  std_logic;" > outfile
  print "       regex_in   : in  std_logic_vector(7 DOWNTO 0);" > outfile
  print "       running    : out std_logic;" > outfile
  print "       accepting  : out std_logic;" > outfile
  print "       reseting   : out std_logic);" > outfile
  print "end regex_fsm" n ";" > outfile
  print "" > outfile
  print "" > outfile
  print "architecture regex_arch" n " of regex_fsm" n " is" > outfile
  print "" > outfile
  printf "  type states is (" > outfile

########################################
# put all the states in the state list #
########################################

  for(x = 0; x < num_states; x++)
  {
    if( x == (num_states - 1) )
      printf "s" x > outfile
    else
      printf "s" x ", " > outfile

    # don't want to many states on 1 line...
    # don't know how many columns the VHDL
    # compiler can handle
    if( x % 15 == 14 )
    {
      print "" > outfile
      printf "                  " > outfile
    }
  }

  print ");" > outfile
  print "" > outfile


  print "  signal state     : states := s0;" > outfile
```

```
 print "  signal nxt_state : states := s0;" > outfile
 print "" > outfile
 print "" > outfile
 print "begin" > outfile


 print "  next_state: process (clk)" > outfile
 print "  begin" > outfile
 print "    if (clk'event and clk = '1') then" > outfile
 print "      if (reset_l = '0') then " > outfile
 print "        state <= s0;" > outfile
 print "      elsif (regex_en = '1') then" > outfile
 print "        state <= nxt_state;" > outfile
 print "      end if;" > outfile
 print "    end if;" > outfile
 print "  end process next_state;" > outfile
 print "" > outfile
 print "" > outfile


################################
# insert sensitivity list here #
################################
  print "  state_trans: process (state, regex_in, regex_en)" > outfile
  print "  begin" > outfile
  print "    nxt_state <= state;" > outfile
  print "    if (regex_en = '1') then" > outfile
  print "" > outfile
  print "      case state is" > outfile
  print "" > outfile

############################################
# determine next states from parsed input #
############################################

  FS = ":"
  for(current_state = 0; current_state < num_states; current_state++)
  {
    else_checker = -1
    for( x = 1; x < 128; x++)   # start at one so a '.' in our regex
    {                           # will not include the NUL character
      nxt_state = yy_nxt[yy_rmap[current_state], yy_cmap[x]]
      if( nxt_state != -1 )
      {
        if(else_checker == current_state)
          printf "                        ELS" > outfile
        else
          printf "        when s" current_state " => " > outfile
        printf "if (regex_in = " > outfile

        while(getline < "characterSet")
        {
          if ( $1 == x )
          {
            if (x != 58)
              printf $3 > outfile
            # we need a special case for semicolon since it is the
            # Field Separator
            else
              printf $4 > outfile
            break
```

```
          }
        }
        close ( "characterSet" )
        if (x != 58)
          print ") then   -- " $2 > outfile
        else
          print ") then   -- ':'" > outfile
        print "                        nxt_state <= s" nxt_state ";" >
outfile
        else_checker = current_state
      }
    }

    if( else_checker == -1 )
      print "        when s" current_state " => nxt_state <= s0;" >
outfile
    else
    {
      print "                    else" > outfile
      print "                      nxt_state <= s0;" > outfile
      print "                    end if;" > outfile
    }
    print "" > outfile
  }

  print "      end case;" > outfile
  print "    end if;" > outfile
  print "  end process state_trans;" > outfile

  print "----------------------------" > outfile
  print "-- CONCURRENT STATEMENTS --" > outfile
  print "----------------------------" > outfile
  print "" > outfile

###########################################
# determine concurrent statements needed #
###########################################

  more = 0
  for(x = 0; x < num_states; x++)
  {
    if( yy_acpt[x] == 1 )
    {
      if ( more == 0 )
        printf "  accepting <= '1' when nxt_state = s" x > outfile
      else
        printf " or nxt_state = s" x > outfile

      more++

      if( more%10 == 9 )
      {
        print "" > outfile
        printf "                  " > outfile
      }

    }
  }
  print " else '0';" > outfile
  print "  reseting  <= '1' when (state /= s0) and (nxt_state  = s0)
else '0';" > outfile
```

```
  print "  running   <= '1' when state /= s0 else '0';" > outfile
  print "" > outfile
  print "end regex_arch" n ";" > outfile

}
```

ReplaceBufGen

```
#!/bin/gawk -f

BEGIN {

  outfile = "replace_buf" n ".vhd"
  len = length(replacement)
  printf "-- This code was generated on " > outfile
  format = "%a %b %e %H:%M:%S %Z %Y"
  print strftime(format) > outfile
  print "-- by a GAWK script created by James Moscola" > outfile
  print "-- The buffer replaces with the word: " replacement > outfile
  print "" > outfile
  print "library ieee;" > outfile
  print "use ieee.std_logic_1164.all;" > outfile
  print "use ieee.std_logic_arith.all;" > outfile
  print "" > outfile
  print "" > outfile
  print "entity replace_buf" n " is" > outfile
  print "  port(clk               : in  std_logic;" > outfile
  print "       reset_l           : in  std_logic;" > outfile
  print "       start_replacing : in  std_logic;" > outfile
  print "       done_replacing  : out std_logic;" > outfile
  print "       new_char          : out std_logic_vector(7 DOWNTO 0));" >
outfile
  print "end replace_buf" n ";" > outfile
  print "" > outfile
  print "" > outfile
  print "architecture replacement_arch" n " of replace_buf" n " is" >
outfile
  print "" > outfile
  print "  type states is (idle, replace);" > outfile
  print "  signal state     : states := idle;" > outfile
  print "" > outfile
  print "  signal cntr : integer := 0;" > outfile
  print "" > outfile

  FS = ":"
  if (len != 1)
  {
    print "  type my_array is array(0 TO "len-1") of std_logic_vector(7
DOWNTO 0);" > outfile
    print "  constant replacement : my_array := (" > outfile
    split(replacement,char_array,"")
    for (x=1; x<=len; x++) {
      char = "'" char_array[x] "'"
      while(getline < "characterSet")
      {
        if ($2 == char)
        {
          if(x != len)
            print "                                       " $3 ",  -- "
char > outfile
          else
            print "                                       " $3 "); -- "
char > outfile
          break
        }
      }
      close("characterSet")
    }
  }
```

```
  else
  {
    printf "  constant replacement : std_logic_vector(7 DOWNTO 0) := " >
outfile
    char = "'" replacement "'"
    while(getline < "characterSet")
    {
      if ($2 == char)
      {
        print $3 "; -- " char > outfile
        break
      }
    }
    close("characterSet")
  }

  print "" > outfile
  print "" > outfile
  print "begin" > outfile
  print "" > outfile
  print "  state_machine: process (clk)" > outfile
  print "  begin" > outfile
  print "    if (clk'event and clk = '1') then" > outfile
  print "      if (reset_l = '0') then" > outfile
  print "        state <= idle;" > outfile
  print "      else" > outfile
  print "        case state is" > outfile
  print "          when idle    => if start_replacing = '1' then" >
outfile
  print "                            state <= replace;" > outfile
  print "                            cntr  <= 0;" > outfile
  print "                          end if;" > outfile
  print "          when replace => if cntr = " len-1 " then" > outfile
  print "                            state <= idle;" > outfile
  print "                            cntr  <= 0;" > outfile
  print "                          else" > outfile
  print "                            cntr <= cntr + 1;" > outfile
  print "                          end if;" > outfile
  print "        end case;" > outfile
  print "      end if;" > outfile
  print "    end if;" > outfile
  print "  end process state_machine;" > outfile
  print "" > outfile
  print "" > outfile

  if (len != 1) {
    print "  new_char <= replacement(cntr);" > outfile
    print "  done_replacing <= '1' when cntr  = " len - 1 " else '0';" >
outfile
  }
  else {
    print "  new_char <= replacement;" > outfile
    print "  done_replacing <= '1' when state = replace else '0';" >
outfile
  }

  print "" > outfile
  print "end replacement_arch" n ";" > outfile

}
```

regex_app.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity regex_app is
  port(clk              : in  std_logic;
       reset_l          : in  std_logic;
       enable_l         : in  std_logic;
       ready_l          : out std_logic;

       dataEn_out_appl  : in  std_logic;
       d_out_appl       : in  std_logic_vector(31 downto 0);
       sof_out_appl     : in  std_logic;
       eof_out_appl     : in  std_logic;
       sod_out_appl     : in  std_logic;
       tca_appl_in      : in  std_logic;

       dataEn_appl_in   : out std_logic;
       d_appl_in        : out std_logic_vector(31 downto 0);
       sof_appl_in      : out std_logic;
       eof_appl_in      : out std_logic;
       sod_appl_in      : out std_logic;
       tca_out_appl     : out std_logic);
end regex_app;


architecture regex_app_arch of regex_app is

  signal dataEn_in              : std_logic;
  signal data_in                : std_logic_vector(31 downto 0);
  signal sof_in                 : std_logic;
  signal eof_in                 : std_logic;
  signal sod_in                 : std_logic;
  signal tca_in                 : std_logic;
  signal dataEn_out             : std_logic;
  signal data_out               : std_logic_vector(31 downto 0);
  signal sof_out                : std_logic;
  signal eof_out                : std_logic;
  signal sod_out                : std_logic;
  signal tca_out                : std_logic;

-- signals for machine #1
  signal ready_l1          : std_logic;

  signal regex_en1         : std_logic;
  signal regex_in1         : std_logic_vector(7 downto 0);
  signal running1          : std_logic;
  signal accepting1        : std_logic;
  signal reseting1         : std_logic;

  signal start_replacing1  : std_logic;
  signal done_replacing1   : std_logic;
  signal new_char1         : std_logic_vector(7 downto 0);



  component regex_fsm1
    port(clk             : in  std_logic;
         reset_l         : in  std_logic;
         regex_en        : in  std_logic;
         regex_in        : in  std_logic_vector(7 downto 0);
```

```
        running          : out std_logic;
        accepting        : out std_logic;
        reseting         : out std_logic);
  end component;


  component replace_buf1
    port(clk              : in  std_logic;
        reset_l          : in  std_logic;
        start_replacing  : in  std_logic;
        done_replacing   : out std_logic;
        new_char         : out std_logic_vector(7 downto 0));
  end component;


  component controller
    port(clk              : in  std_logic;
        reset_l          : in  std_logic;
        enable_l         : in  std_logic;
        ready_l          : out std_logic;

        dataEn_out_appl  : in  std_logic;
        d_out_appl       : in  std_logic_vector(31 downto 0);
        sof_out_appl     : in  std_logic;
        eof_out_appl     : in  std_logic;
        sod_out_appl     : in  std_logic;
        tca_appl_in      : in  std_logic;

        dataEn_appl_in   : out std_logic;
        d_appl_in        : out std_logic_vector(31 downto 0);
        sof_appl_in      : out std_logic;
        eof_appl_in      : out std_logic;
        sod_appl_in      : out std_logic;
        tca_out_appl     : out std_logic;

        regex_en         : out std_logic;
        regex_in         : out std_logic_vector(7 downto 0);
        running          : in  std_logic;
        accepting        : in  std_logic;
        reseting         : in  std_logic;

        start_replacing  : out std_logic;
        done_replacing   : in  std_logic;
        new_char         : in  std_logic_vector(7 downto 0));
  end component;


begin

  regular_expression_machine1 : regex_fsm1
  port map(clk                => clk,
          reset_l            => reset_l,
          regex_en           => regex_en1,
          regex_in           => regex_in1,
          running            => running1,
          accepting          => accepting1,
          reseting           => reseting1);
```

```
  replacement_buffer1 : replace_buf1
  port map(clk                => clk,
           reset_l            => reset_l,
           start_replacing => start_replacing1,
           done_replacing  => done_replacing1,
           new_char           => new_char1);


  controller1 : controller
  port map(clk                => clk,
           reset_l            => reset_l,
           enable_l           => enable_l,
           ready_l            => ready_l1,

           dataEn_out_appl => dataEn_in,
           d_out_appl         -> data_in,
           sof_out_appl       => sof_in,
           eof_out_appl       => eof_in,
           sod_out_appl       => sod_in,
           tca_appl_in        => tca_in,        -- take in from downstream
mod

           dataEn_appl_in  => dataEn_out,    -- send to downstream mod
           d_appl_in          => data_out,
           sof_appl_in        => sof_out,
           eof_appl_in        => eof_out,
           sod_appl_in        => sod_out,
           tca_out_appl       => tca_out,

-- SIGNALS FOR SEARCH AND REPLACE
           regex_en           => regex_en1,
           regex_in           => regex_in1,
           running            => running1,
           accepting          => accepting1,
           reseting           => reseting1,

           start_replacing => start_replacing1,
           done_replacing  => done_replacing1,
           new_char           => new_char1);


--------------------------------------------------------------------------
-----

  flop_signals: process (clk)
  begin
    if clk'event and clk = '1' then
      dataEn_in          <= dataEn_out_appl;
      data_in            <= d_out_appl;
      sof_in             <= sof_out_appl;
      eof_in             <= eof_out_appl;
      sod_in             <= sod_out_appl;
      tca_in             <= tca_appl_in;

      dataEn_appl_in <= dataEn_out;
      d_appl_in          <= data_out;
      sof_appl_in        <= sof_out;
      eof_appl_in        <= eof_out;
      sod_appl_in        <= sod_out;
      tca_out_appl       <= tca_out;
```

```
    end if;
  end process flop_signals;

  ready_l <= ready_l1;

end regex_app_arch;
```

replace_buf1.vhd

```
-- This code was generated on Tue Jan  13:27:24  2002
-- by a GAWK script created by James Moscola
-- The buffer replaces with the word: this is a test

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;


entity replace_buf1 is
  port(clk              : in  std_logic;
       reset_l          : in  std_logic;
       start_replacing  : in  std_logic;
       done_replacing   : out std_logic;
       new_char         : out std_logic_vector(7 DOWNTO 0));
end replace_buf1;


architecture replacement_arch1 of replace_buf1 is

  type states is (idle, replace);
  signal state     : states := idle;

  signal cntr : integer := 0;

  type my_array is array(0 TO 13) of std_logic_vector(7 DOWNTO 0);
  constant replacement : my_array := (
                                       "01110100",  -- 't'
                                       "01101000",  -- 'h'
                                       "01101001",  -- 'i'
                                       "01110011",  -- 's'
                                       "00100000",  -- ' '
                                       "01101001",  -- 'i'
                                       "01110011",  -- 's'
                                       "00100000",  -- ' '
                                       "01100001",  -- 'a'
                                       "00100000",  -- ' '
                                       "01110100",  -- 't'
                                       "01100101",  -- 'e'
                                       "01110011",  -- 's'
                                       "01110100"); -- 't'


begin

  state_machine: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (reset_l = '0') then
        state <= idle;
      else
        case state is
          when idle    => if start_replacing = '1' then
                             state <= replace;
                             cntr  <= 0;
                           end if;
          when replace => if cntr = 13 then
                             state <= idle;
                             cntr  <= 0;
                           else
                             cntr <= cntr + 1;
```

```
                          end if;
          end case;
        end if;
      end if;
    end process state_machine;


    new_char <= replacement(cntr);
    done_replacing <= '1' when cntr  = 13 else '0';

  end replacement_arch1;
```

controller.vhd

```
-- increment headers and trailers by 4
-- fix reset on replace_buf(DONE)  and maybe in other places.
-- enable_l... don't give ready until buffer is empty
-- give replace_buf ability to do a null replacement ''
-- remove byte_ptr and use prev_addra(1 downto 0) instead

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity controller is
  port(clk             : in  std_logic;
       reset_l         : in  std_logic;
       enable_l        : in  std_logic;
       ready_l         : out std_logic;

       dataen_out_appl : in  std_logic;
       d_out_appl      : in  std_logic_vector(31 downto 0);
       sof_out_appl    : in  std_logic;
       eof_out_appl    : in  std_logic;
       sod_out_appl    : in  std_logic;
       tca_appl_in     : in  std_logic;

       dataen_appl_in  : out std_logic;
       d_appl_in       : out std_logic_vector(31 downto 0);
       sof_appl_in     : out std_logic;
       eof_appl_in     : out std_logic;
       sod_appl_in     : out std_logic;
       tca_out_appl    : out std_logic;

       regex_en        : out std_logic;
       regex_in        : out std_logic_vector(7 downto 0);
       running         : in  std_logic;
       accepting       : in  std_logic;
       reseting        : in  std_logic;

       start_replacing : out std_logic;
       done_replacing  : in  std_logic;
       new_char        : in  std_logic_vector(7 downto 0));
end controller;


architecture controller_arch of controller is

  -----------------
  -- signal list --
  -----------------

  signal prev_addra      : std_logic_vector(10 downto 0)  :=  (others =>
'0');
  signal accept_ptr      : std_logic_vector(10 downto 0)  :=  (others =>
'0');
  signal cntr_addra_out  : std_logic_vector(10 downto 0)  :=  (others =>
'0');
  signal out_ptr         : std_logic_vector(10 downto 0)  :=  (others =>
'0');
  signal back_ptr        : std_logic_vector(10 downto 0)  :=  (others =>
'0');
  signal cntr_addrb_in   : std_logic_vector( 8 downto 0)  :=  (others =>
```

```
'0');
  signal byte_ptr          : std_logic_vector( 1 downto 0)  :=  "00";
  signal byte_ptr2         : std_logic_vector( 1 downto 0)  :=  "00";

  signal web               : std_logic;
  signal char_buf_out      : std_logic_vector(32 downto 0);
  signal out_buf_out       : std_logic_vector(32 downto 0);
  signal old_char          : std_logic_vector( 7 downto 0);
  signal old_char_en       : std_logic := '0';

  signal regex_enable      : std_logic;
  signal start_replacing_out : std_logic;

  signal wrd_bldr_in_en    : std_logic := '0';
  signal wrd_bldr_in       : std_logic_vector(7 downto 0);
  signal wrd_bldr_cntr     : std_logic_vector(1 downto 0);
  signal wrd_bldr_out      : std_logic_vector(31 downto 0);
  signal wrd_bldr_out_en : std_logic;

  signal char_buf_addra    : std_logic_vector(8 downto 0);

  signal accepted          : std_logic := '0';
  signal buf_full          : std_logic := '0';
  signal fifo_full         : std_logic := '0';
  signal fifo_empty        : std_logic := '1';
  signal out_buf_en        : std_logic := '0';


  subtype fifo_loc is std_logic_vector(10 downto 0);
  type fifo is array(0 to 7) of fifo_loc;
  signal match_begin_fifo     : fifo := (others => (others => '0'));
  signal match_end_fifo       : fifo := (others => (others => '0'));
  signal fifo_rd_ptr  : std_logic_vector(2 downto 0)  := "000";
  signal fifo_wr_ptr  : std_logic_vector(2 downto 0)  := "000";

  signal dib : std_logic_vector(32 downto 0);

  type pkt_states          is (atm_hdr, len, hdr, data, flush, trailer,
replace);
  signal fsm_state         : pkt_states := atm_hdr;
  signal out_state         : pkt_states := atm_hdr;

  signal pkt_len           : std_logic_vector(15 downto 0)  := (others =>
'0');
  signal pkt_len_cnt       : std_logic_vector(15 downto 0)  := (others =>
'0');
  signal pkt_len_cnt2      : std_logic_vector(15 downto 0)  := (others =>
'0');
  signal pkt_len_out       : std_logic_vector(15 downto 0)  := (others =>
'0');
  signal trail_cntr        : std_logic_vector (2 downto 0)  := (others =>
'1');


  alias  is_sof_address   : std_logic is char_buf_out(32);
  alias  is_sof_address2  : std_logic is out_buf_out(32);



  ----------------
```

```
  -- components --
  ----------------

  component character_buf
    port(addra             : in  std_logic_vector( 8 downto 0);
         clka              : in  std_logic;
         addrb             : in  std_logic_vector( 8 downto 0);
         clkb              : in  std_logic;
         dib               : in  std_logic_vector(32 downto 0);
         web               : in  std_logic;
         doa               : out std_logic_vector(32 downto 0));
  end component;


  component wrd_bldr
  port(clk                 : in  std_logic;
       reset_l             : in  std_logic;
       wrd_bldr_in_en      : in  std_logic;
       wrd_bldr_in         : in  std_logic_vector( 7 downto 0);
       wrd_bldr_cntr       : out std_logic_vector( 1 downto 0);
       wrd_bldr_out        : out std_logic_vector(31 downto 0);
       wrd_bldr_out_en     : out std_logic);
  end component;


begin

  ---------------------
  -- structural ties --
  ---------------------

  character_buffer : character_buf
  port map(addra           => char_buf_addra,
           clka            => clk,
           addrb           => cntr_addrb_in(8 downto 0),
           clkb            => clk,
           dib             => dib,
           web             => web,
           doa             => char_buf_out);


  output_buffer : character_buf
  port map(addra           => out_ptr(10 downto 2),
           clka            => clk,
           addrb           => cntr_addrb_in(8 downto 0),
           clkb            => clk,
           dib             => dib,
           web             => web,
           doa             => out_buf_out);


  word_builder : wrd_bldr
  port map(clk             => clk,
           reset_l         => reset_l,
           wrd_bldr_in_en  => wrd_bldr_in_en,
           wrd_bldr_in     => wrd_bldr_in,
           wrd_bldr_cntr   => wrd_bldr_cntr,
           wrd_bldr_out    => wrd_bldr_out,
           wrd_bldr_out_en => wrd_bldr_out_en);
```

```
--------------------------------------------------------------------------
-
  -- the following process controls the pointer for the input side of
the
  -- dual-port memory
--------------------------------------------------------------------------
-

  count_address_in: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (sof_out_appl = '1' or dataen_out_appl = '1') then
        cntr_addrb_in <= cntr_addrb_in + 1;

        if ((cntr_addrb_in + 92) = out_ptr(10 downto 2)) then
          buf_full <= '1';
        end if;
      end if;

      if ((cntr_addrb_in = (cntr_addra_out(10 downto 2) + 1)) and
          (out_ptr(10 downto 2) /= (cntr_addrb_in + 1))) then
        buf_full <= '0';
      end if;
    end if;
  end process count_address_in;



  ------------------------------------------------------------
  -- the following processes are for controlling the input,
  -- and the enable to the regular expression machine
  ------------------------------------------------------------

  regex_input_machine: process (clk)
  begin
    if clk'event and clk = '1' then
      if reset_l = '0' then
        fsm_state <= atm_hdr;
        regex_enable <= '0';
      else
        case fsm_state is
          when atm_hdr => if cntr_addra_out /= cntr_addrb_in & "00" then
                            cntr_addra_out <= cntr_addra_out + 1;
                            if is_sof_address = '1' and byte_ptr = "11"
then
                              fsm_state <= len;
                            end if;
                          end if;

          when len     => if cntr_addra_out /= cntr_addrb_in & "00" then
                            cntr_addra_out <= cntr_addra_out + 1;
                            pkt_len      <= char_buf_out(15 downto 0);
                            pkt_len_cnt  <= char_buf_out(15 downto 0) -
1;
                            if is_sof_address = '0' or byte_ptr /= "11"
then   -- this may be wrong
                              fsm_state <= hdr;
```

```
                                  end if;
                                end if;

             when hdr       => if cntr_addra_out /= cntr_addrb_in & "00" then
                                  cntr_addra_out <= cntr_addra_out + 1;
                                  pkt_len_cnt <= pkt_len_cnt - 1;
                                end if;

                                if is_sof_address = '1' and byte_ptr = "11"
then
                                  fsm_state <= len;
                                elsif pkt_len_cnt + x"1B" = pkt_len then
                                  fsm_state <= data;
                                  regex_enable <= '1';
                                end if;

             when data      => if reseting = '1' then
                                  if accepted = '1' then
                                    cntr_addra_out <= accept_ptr + 1;
                                    regex_enable <= '0';
                                    pkt_len_cnt <= pkt_len_cnt +
(cntr_addra_out - (accept_ptr + 1));
                                  else
                                    cntr_addra_out <= back_ptr + 1;
                                    regex_enable <= '0';
                                    pkt_len_cnt <= pkt_len_cnt +
(cntr_addra_out - (back_ptr + 1));
                                  end if;
                                elsif fifo_full = '0' then
                                  if cntr_addra_out /= cntr_addrb_in & "00"
then
                                    cntr_addra_out <= cntr_addra_out + 1;
                                    regex_enable <= '1';
                                    pkt_len_cnt <= pkt_len_cnt - 1;
                                  else
                                    regex_enable <= '0';
                                  end if;
                                end if;

                                if is_sof_address = '1' then
                                  regex_enable <= '0';
                                  if byte_ptr = "11" then
                                    fsm_state <= len;
                                  end if;
                                end if;

                                if pkt_len_cnt = x"00" and running = '0' and
byte_ptr = "00" then
                                  fsm_state <= trailer;
                                  regex_enable <= '0';
                                end if;

             when trailer => if cntr_addra_out /= cntr_addrb_in & "00" then
                                  cntr_addra_out <= cntr_addra_out + 1;
                                  pkt_len_cnt <= pkt_len_cnt - 1;
                                end if;

                                if is_sof_address = '1' and byte_ptr = "11"
then
                                  fsm_state <= len;
                                elsif pkt_len_cnt = x"FFF9" then
```

```
                                fsm_state <= atm_hdr;
                            end if;

            when others  => null;
          end case;
        end if;
      end if;
    end process regex_input_machine;



-----------------------------------------------------------------------
--------
-- these processes take care of a little book-keeping.
-----------------------------------------------------------------------
--------

  previous_address: process (clk)
  begin
    if (clk'event and clk = '1') then
      prev_addra <= cntr_addra_out;
    end if;
  end process previous_address;


  backtrack: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (running = '0') then
        back_ptr <= prev_addra;
      end if;
    end if;
  end process backtrack;


  made_match: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (accepting = '1' and regex_enable = '1') then
        accepted <= '1';
        accept_ptr <= prev_addra;
      elsif (reseting = '1') then
        accepted <= '0';
      end if;
    end if;
  end process made_match;



-----------------------------------------------------------------------
-
-----------------------------------------------------------------------
-
-- the following processes are for outputting the data
-----------------------------------------------------------------------
-
```

```
output_fsm_machine: process (clk)
begin
  if clk'event and clk = '1' then
    if reset_l = '0' then
      out_state           <= atm_hdr;
      old_char_en         <= '0';
      start_replacing_out <= '0';
    else
      old_char_en         <= '0';
      start_replacing_out <= '0';
      case out_state is
        when atm_hdr => if out_ptr /= back_ptr then
                          out_ptr     <= out_ptr + 1;
                          old_char_en <= '1';
                          if is_sof_address2 = '1' and byte_ptr2 =
"11" then
                            out_state <= len;
                          end if;
                        end if;
                        trail_cntr <= "111";

        when len     => if out_ptr /= back_ptr then
                          out_ptr     <= out_ptr + 1;
                          old_char_en <= '1';
                          if is_sof_address2 = '0' or byte_ptr2 /=
"11" then  ·
                            out_state <= hdr;
                          end if;
                        end if;

        when hdr     => if out_ptr /= back_ptr then
                          out_ptr     <= out_ptr + 1;
                          old_char_en <= '1';
                        end if;

                        if is_sof_address2 = '1' and byte_ptr2 = "11"
then
                          out_state <= len;
                        elsif pkt_len_cnt2 + x"1B" = pkt_len_out then
                          out_state <= data;
                        end if;

        when data     => if out_ptr /= back_ptr then
                          out_ptr     <= out_ptr + 1;
                          old_char_en <= '1';
                          if is_sof_address2 = '1' and byte_ptr2 =
"11" then
                            out_state   <= len;
                          elsif match_begin_fifo(conv_integer
(fifo_rd_ptr)) = out_ptr and fifo_empty = '0' then
                            out_state   <= replace;
                            old_char_en <= '0';
                            out_ptr     <= match_end_fifo(conv_integer
(fifo_rd_ptr));
                            fifo_rd_ptr <= fifo_rd_ptr + 1;
                            start_replacing_out <= '1';
                          else
                            if pkt_len_cnt2 = x"01" then
                              if wrd_bldr_cntr = "11" then
                                out_state <= trailer;
```

```
                        else
                          out_ptr <= out_ptr;
                          old_char_en <= '0';
                          out_state <= flush;
                        end if;
                      end if;
                    end if;
                  end if;

          when flush    => if wrd_bldr_cntr = "11" then
                    out_ptr <= out_ptr + 1;
                    old_char_en <= '1';
                    out_state <= trailer;
                  end if;

          when trailer => if out_ptr /= back_ptr then
                    out_ptr      <= out_ptr + 1;
                    old_char_en <= '1';
                  end if;

                  if trail_cntr = "000" then
                    out_state  <= atm_hdr;
                  else
                    trail_cntr <= trail_cntr - 1;
                  end if;

          when replace => if done_replacing = '1' then
                    if pkt_len_cnt2 = x"00" then
                      if wrd_bldr_cntr = "11" then
                        out_state <= trailer;
                      else
                        out_state <= flush;
                      end if;
                    else
                      out_state <= data;
                    end if;
                  end if;
        end case;
      end if;
    end if;
  end process output_fsm_machine;


  -- This process controls the pkt_len_cnt2 register.  This register
  -- keeps track of how many bytes are left to output in a packet.
  packet_length_counter2: process (clk)
  begin
    if clk'event and clk = '1' then
      if out_state = len then
        pkt_len_cnt2 <= out_buf_out(15 downto 0) - 1;
      elsif start_replacing_out = '1' then
        pkt_len_cnt2 <= pkt_len_cnt2  -
                       (match_end_fifo(conv_integer(fifo_rd_ptr - 1)) -
                        match_begin_fifo(conv_integer(fifo_rd_ptr -
1)));
      elsif old_char_en = '1' then
        pkt_len_cnt2 <= pkt_len_cnt2 - 1;
      end if;
    end if;
```

```
  end process packet_length_counter2;



  -- This process controls the pkt_len_out register.  This register
  -- keeps track of the length of the current packet being output.
  packet_length_out: process (clk)
  begin
    if clk'event and clk = '1' then
      if out_state = len then
        pkt_len_out <= out_buf_out(15 downto 0);
      elsif start_replacing_out = '1' then
        pkt_len_out <= pkt_len_out -
                          (match_end_fifo(conv_integer(fifo_rd_ptr - 1))
-
                           match_begin_fifo(conv_integer(fifo_rd_ptr -
1)));
      elsif start_replacing_out = '0' and out_state = replace then
        pkt_len_out <= pkt_len_out + 1;
      end if;
    end if;
  end process packet_length_out;



  -- Here are the control signals that are sent to the UDP Wrapper to
indicate:
  -- start of frame, start of datagram, and end of frame.
  sof_appl_in <= '1' when out_state = len else '0';
  sod_appl_in <= '1' when (pkt_len_cnt2 = pkt_len_out - x"18") else '0';
  eof_appl_in <= '1' when out_state = trailer and wrd_bldr_out_en = '1'
and pkt_len_cnt2 < "011" else '0';




------------------------------------------------------------
------------------------------------------------------------
-- the following processes control the found and replace fifo
------------------------------------------------------------

  fifo_write_pointer: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (reseting = '1' and accepted = '1') then
        match_begin_fifo(conv_integer(fifo_wr_ptr)) <= back_ptr;
        match_end_fifo(conv_integer(fifo_wr_ptr))   <= accept_ptr + 1;
-- stores the next position after the
        fifo_wr_ptr <= fifo_wr_ptr + 1;
-- replacement is complete
      end if;
    end if;
  end process fifo_write_pointer;


  fifo_full  <= '1' when fifo_wr_ptr + 1 = fifo_rd_ptr else '0';
  fifo_empty <= '1' when fifo_rd_ptr = fifo_wr_ptr else '0';
```

```
----------------------------------------------------------------

  byte_counter: process (clk)
  begin
    if (clk'event and clk = '1') then
      byte_ptr <= cntr_addra_out(1 downto 0);
    end if;
  end process byte_counter;


  byte_counter2: process (clk)
  begin
    if (clk'event and clk = '1') then
      byte_ptr2 <= out_ptr(1 downto 0);
    end if;
  end process byte_counter2;


  ---------------------------
  -- concurrent statements --
  ---------------------------

  ready_l <= not enable_l;
  regex_en         <= regex_enable;
  start_replacing <= start_replacing_out;


  web <= '1' when sof_out_appl = '1' or dataEn_out_appl = '1' else '0';
  dib <= sof_out_appl & d_out_appl;


  char_buf_addra <= cntr_addra_out(10 downto 2);


  regex_in <= "00000000"                     when pkt_len_cnt = x"00" else
              char_buf_out(31 downto 24) when byte_ptr     =  "00" else
              char_buf_out(23 downto 16) when byte_ptr     =  "01" else
              char_buf_out(15 downto  8) when byte_ptr     =  "10" else
              char_buf_out( 7 downto  0);


  old_char <= x"00" when out_state = flush  and old_char_en /= '1' else
                  pkt_len_out(15 downto  8) when trail_cntr = "001" and
byte_ptr2 = "10" else
                  pkt_len_out( 7 downto  0) when trail_cntr = "000" and
byte_ptr2 = "11" else
                  out_buf_out(31 downto 24) when byte_ptr2 = "00" else
                  out_buf_out(23 downto 16) when byte_ptr2 = "01" else
                  out_buf_out(15 downto  8) when byte_ptr2 = "10" else
                  out_buf_out( 7 downto  0);


  wrd_bldr_in    <= new_char when (out_state = replace and
start_replacing_out = '0') else old_char;
  wrd_bldr_in_en <= '1' when out_state = flush or old_char_en = '1' or
                         (out_state = replace and
start_replacing_out = '0') else '0';
```

```
  dataen_appl_in <= '1' when wrd_bldr_out_en = '1' and out_state /= len
else '0';
  d_appl_in <= wrd_bldr_out;

  tca_out_appl    <= '0' when tca_appl_in = '0' or buf_full = '1' else
'1';

end controller_arch;
```

wrd_bldr.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;


entity wrd_bldr is
  port(clk             : in  std_logic;
       reset_l         : in  std_logic;
       wrd_bldr_in_en  : in  std_logic;
       wrd_bldr_in     : in  std_logic_vector(7 downto 0);
       wrd_bldr_cntr   : out std_logic_vector( 1 downto 0);
       wrd_bldr_out    : out std_logic_vector(31 downto 0);
       wrd_bldr_out_en : out std_logic);
end wrd_bldr;


architecture behavioral of wrd_bldr is

  type rammemory is array(2 downto 0) of std_logic_vector(7 downto 0);
  signal ram : rammemory;
  signal ptr : integer range 0 to 3 := 0;

begin

  build_words: process(clk)
  begin
    if clk'event and clk= '1' then
      wrd_bldr_out_en <= '0';

      if reset_l = '0' then
        ptr <= 0;
      else
        if (wrd_bldr_in_en = '1' and ptr = 3) then
          wrd_bldr_out_en <= '1';
          wrd_bldr_out(31 downto 24) <= ram(0);
          wrd_bldr_out(23 downto 16) <= ram(1);
          wrd_bldr_out(15 downto  8) <= ram(2);
          wrd_bldr_out( 7 downto  0) <= wrd_bldr_in(7 downto 0);
          ptr <= 0;
        elsif wrd_bldr_in_en = '1' then
          ram(ptr) <= wrd_bldr_in;
          ptr <= ptr + 1;
        end if;
      end if;
    end if;
  end process build_words;

  wrd_bldr_cntr <=  conv_std_logic_vector(ptr,2);

end behavioral;
```

```
-- This code was generated on Tue Jan  13:27:24  2002
-- by a GAWK script created by James Moscola
-- The buffer replaces with the word: this is a test

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;


entity replace_buf1 is
  port(clk              : in  std_logic;
       reset_l          : in  std_logic;
       start_replacing  : in  std_logic;
       done_replacing   : out std_logic;
       new_char         : out std_logic_vector(7 DOWNTO 0));
end replace_buf1;


architecture replacement_arch1 of replace_buf1 is

  type states is (idle, replace);
  signal state     : states := idle;

  signal cntr : integer := 0;

  type my_array is array(0 TO 13) of std_logic_vector(7 DOWNTO 0);
  constant replacement : my_array := (
                                       "01110100",  -- 't'
                                       "01101000",  -- 'h'
                                       "01101001",  -- 'i'
                                       "01110011",  -- 's'
                                       "00100000",  -- ' '
                                       "01101001",  -- 'i'
                                       "01110011",  -- 's'
                                       "00100000",  -- ' '
                                       "01100001",  -- 'a'
                                       "00100000",  -- ' '
                                       "01110100",  -- 't'
                                       "01100101",  -- 'e'
                                       "01110011",  -- 's'
                                       "01110100"); -- 't'


begin

  state_machine: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (reset_l = '0') then
        state <= idle;
      else
        case state is
          when idle    => if start_replacing = '1' then
                            state <= replace;
                            cntr  <= 0;
                          end if;
          when replace => if cntr = 13 then
                            state <= idle;
                            cntr  <= 0;
                          else
                            cntr <= cntr + 1;
```

```
                          end if;
        end case;
      end if;
    end if;
  end process state_machine;


  new_char <= replacement(cntr);
  done_replacing <= '1' when cntr  = 13 else '0';

end replacement_arch1;
```

character_buf.vhd

```
library ieee;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

-- synopsys translate_off
library XilinxCoreLib;
-- synopsys translate_on


entity character_buf is
  port(addra: in std_logic_vector(8 downto 0);
       clka: in std_logic;
       addrb: in std_logic_vector(8 downto 0);
       clkb: in std_logic;
       dib: in std_logic_vector(32 downto 0);
       web: in std_logic;
       doa: OUT std_logic_vector(32 downto 0));
end character_buf;


architecture structure of character_buf is

component character_buf
      port (
      addra: IN std_logic_VECTOR(8 downto 0);
      clka: IN std_logic;
      addrb: IN std_logic_VECTOR(8 downto 0);
      clkb: IN std_logic;
      dib: IN std_logic_VECTOR(32 downto 0);
      web: IN std_logic;
      doa: OUT std_logic_VECTOR(32 downto 0));
end component;

-- Synplicity black box declaration
attribute black_box : boolean;
attribute black_box of character_buf: component is true;

-- synopsys translate_off
  for all : character_buf use entity XilinxCoreLib.C_MEM_DP_BLOCK_V1_0
(behavioral)
    generic map(c_depth_b => 512,
                c_depth_a => 512,
                c_has_web => 1,
                c_has_wea => 0,
                c_has_dib => 1,
                c_has_dia => 0,
                c_clka_polarity => 1,
                c_web_polarity => 1,
                c_address_width_b => 9,
                c_address_width_a => 9,
                c_width_b => 33,
                c_width_a => 33,
                c_clkb_polarity => 1,
                c_ena_polarity => 1,
                c_rsta_polarity => 1,
                c_has_rstb => 0,
                c_has_rsta => 0,
                c_read_mif => 0,
                c_enb_polarity => 1,
                c_pipe_stages => 0,
                c_rstb_polarity => 1,
```

```
                c_has_enb => 0,
                c_has_ena => 0,
                c_mem_init_radix => 16,
                c_default_data => "0",
                c_mem_init_file => "character_buf.mif",
                c_has_dob => 0,
                c_generate_mif => 1,
                c_has_doa => 1,
                c_wea_polarity => 1);

-- synopsys translate_on

  begin
    character_buffer : character_buf
    port map(addra => addra,
             clka => clka,
             addrb => addrb,
             clkb => clkb,
             dib => dib,
             web => web,
             doa => doa);

end structure;
```

regex_fsm1.vhd

```
-- This code was generated on Tue Jan  13:27:22  2002
-- by a GAWK script created by James Moscola (4-22-2001)
-- Regular Expression is:    t.*t

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;


entity regex_fsm1 is
  port(clk        : in  std_logic;
       reset_l    : in  std_logic;
       regex_en   : in  std_logic;
       regex_in   : in  std_logic_vector(7 DOWNTO 0);
       running    : out std_logic;
       accepting  : out std_logic;
       reseting   : out std_logic);
end regex_fsm1;


architecture regex_arch1 of regex_fsm1 is

  type states is (s0, s1, s2, s3);

  signal state     : states := s0;
  signal nxt_state : states := s0;


begin
  next_state: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (reset_l = '0') then
        state <= s0;
      elsif (regex_en = '1') then
        state <= nxt_state;
      end if;
    end if;
  end process next_state;


  state_trans: process (state, regex_in, regex_en)
  begin
    nxt_state <= state;
    if (regex_en = '1') then

      case state is

        when s0 => if (regex_in = "01110100") then  -- 't'
                     nxt_state <= s3;
                   else
                     nxt_state <= s0;
                   end if;

        when s1 => nxt_state <= s0;

        when s2 => if (regex_in = "00000001") then  -- SOH
                     nxt_state <= s3;
                   ELSif (regex_in = "00000010") then  -- STX
                     nxt_state <= s3;
                   ELSif (regex_in = "00000011") then  -- ETX
```

```
  nxt_state <= s3;
ELSif (regex_in = "00000100") then  -- EOT
  nxt_state <= s3;
ELSif (regex_in = "00000101") then  -- ENQ
  nxt_state <= s3;
ELSif (regex_in = "00000110") then  -- ACK
  nxt_state <= s3;
ELSif (regex_in = "00000111") then  -- BEL
  nxt_state <= s3;
ELSif (regex_in = "00001000") then  -- BS
  nxt_state <= s3;
ELSif (regex_in = "00001001") then  -- HT
  nxt_state <= s3;
ELSif (regex_in = "00001011") then  -- VT
  nxt_state <= s3;
ELSif (regex_in = "00001100") then  -- FF
  nxt_state <= s3;
ELSif (regex_in = "00001110") then  -- SO
  nxt_state <= s3;
ELSif (regex_in = "00001111") then  -- SI
  nxt_state <= s3;
ELSif (regex_in = "00010000") then  -- DLE
  nxt_state <= s3;
ELSif (regex_in = "00010001") then  -- DC1
  nxt_state <= s3;
ELSif (regex_in = "00010010") then  -- DC2
  nxt_state <= s3;
ELSif (regex_in = "00010011") then  -- DC3
  nxt_state <= s3;
ELSif (regex_in = "00010100") then  -- DC4
  nxt_state <= s3;
ELSif (regex_in = "00010101") then  -- NAK
  nxt_state <= s3;
ELSif (regex_in = "00010110") then  -- SYN
  nxt_state <= s3;
ELSif (regex_in = "00010111") then  -- ETB
  nxt_state <= s3;
ELSif (regex_in = "00011000") then  -- CAN
  nxt_state <= s3;
ELSif (regex_in = "00011001") then  -- EM
  nxt_state <= s3;
ELSif (regex_in = "00011010") then  -- SUB
  nxt_state <= s3;
ELSif (regex_in = "00011011") then  -- ESC
  nxt_state <= s3;
ELSif (regex_in = "00011100") then  -- FSP
  nxt_state <= s3;
ELSif (regex_in = "00011101") then  -- GSP
  nxt_state <= s3;
ELSif (regex_in = "00011110") then  -- RSP
  nxt_state <= s3;
ELSif (regex_in = "00011111") then  -- USP
  nxt_state <= s3;
ELSif (regex_in = "00100000") then  -- ' '
  nxt_state <= s3;
ELSif (regex_in = "00100001") then  -- '!'
  nxt_state <= s3;
ELSif (regex_in = "00100010") then  -- '"'
  nxt_state <= s3;
ELSif (regex_in = "00100011") then  -- '#'
  nxt_state <= s3;
```

```
ELSif (regex_in = "00100100") then  -- '$'
  nxt_state <= s3;
ELSif (regex_in = "00100101") then  -- '%'
  nxt_state <= s3;
ELSif (regex_in = "00100110") then  -- '&'
  nxt_state <= s3;
ELSif (regex_in = "00100111") then  -- '''
  nxt_state <= s3;
ELSif (regex_in = "00101000") then  -- '('
  nxt_state <= s3;
ELSif (regex_in = "00101001") then  -- ')'
  nxt_state <= s3;
ELSif (regex_in = "00101010") then  -- '*'
  nxt_state <= s3;
ELSif (regex_in = "00101011") then  -- '+'
  nxt_state <= s3;
ELSif (regex_in = "00101100") then  -- ','
  nxt_state <= s3;
ELSif (regex_in = "00101101") then  -- '-'
  nxt_state <= s3;
ELSif (regex_in = "00101110") then  -- '.'
  nxt_state <= s3;
ELSif (regex_in = "00101111") then  -- '/'
  nxt_state <= s3;
ELSif (regex_in = "00110000") then  -- '0'
  nxt_state <= s3;
ELSif (regex_in = "00110001") then  -- '1'
  nxt_state <= s3;
ELSif (regex_in = "00110010") then  -- '2'
  nxt_state <= s3;
ELSif (regex_in = "00110011") then  -- '3'
  nxt_state <= s3;
ELSif (regex_in = "00110100") then  -- '4'
  nxt_state <= s3;
ELSif (regex_in = "00110101") then  -- '5'
  nxt_state <= s3;
ELSif (regex_in = "00110110") then  -- '6'
  nxt_state <= s3;
ELSif (regex_in = "00110111") then  -- '7'
  nxt_state <= s3;
ELSif (regex_in = "00111000") then  -- '8'
  nxt_state <= s3;
ELSif (regex_in = "00111001") then  -- '9'
  nxt_state <= s3;
ELSif (regex_in = "00111010") then  -- ':'
  nxt_state <= s3;
ELSif (regex_in = "00111011") then  -- ';'
  nxt_state <= s3;
ELSif (regex_in = "00111100") then  -- '<'
  nxt_state <= s3;
ELSif (regex_in = "00111101") then  -- '='
  nxt_state <= s3;
ELSif (regex_in = "00111110") then  -- '>'
  nxt_state <= s3;
ELSif (regex_in = "00111111") then  -- '?'
  nxt_state <= s3;
ELSif (regex_in = "01000000") then  -- '@'
  nxt_state <= s3;
ELSif (regex_in = "01000001") then  -- 'A'
  nxt_state <= s3;
ELSif (regex_in = "01000010") then  -- 'B'
```

```
    nxt_state <= s3;
  ELSif (regex_in = "01000011") then  -- 'C'
    nxt_state <= s3;
  ELSif (regex_in = "01000100") then  -- 'D'
    nxt_state <= s3;
  ELSif (regex_in = "01000101") then  -- 'E'
    nxt_state <= s3;
  ELSif (regex_in = "01000110") then  -- 'F'
    nxt_state <= s3;
  ELSif (regex_in = "01000111") then  -- 'G'
    nxt_state <= s3;
  ELSif (regex_in = "01001000") then  -- 'H'
    nxt_state <= s3;
  ELSif (regex_in = "01001001") then  -- 'I'
    nxt_state <= s3;
  ELSif (regex_in = "01001010") then  -- 'J'
    nxt_state <= s3;
  ELSif (regex_in = "01001011") then  -- 'K'
    nxt_state <= s3;
  ELSif (regex_in = "01001100") then  -- 'L'
    nxt_state <= s3;
  ELSif (regex_in = "01001101") then  -- 'M'
    nxt_state <= s3;
  ELSif (regex_in = "01001110") then  -- 'N'
    nxt_state <= s3;
  ELSif (regex_in = "01001111") then  -- 'O'
    nxt_state <= s3;
  ELSif (regex_in = "01010000") then  -- 'P'
    nxt_state <= s3;
  ELSif (regex_in = "01010001") then  -- 'Q'
    nxt_state <= s3;
  ELSif (regex_in = "01010010") then  -- 'R'
    nxt_state <= s3;
  ELSif (regex_in = "01010011") then  -- 'S'
    nxt_state <= s3;
  ELSif (regex_in = "01010100") then  -- 'T'
    nxt_state <= s3;
  ELSif (regex_in = "01010101") then  -- 'U'
    nxt_state <= s3;
  ELSif (regex_in = "01010110") then  -- 'V'
    nxt_state <= s3;
  ELSif (regex_in = "01010111") then  -- 'W'
    nxt_state <= s3;
  ELSif (regex_in = "01011000") then  -- 'X'
    nxt_state <= s3;
  ELSif (regex_in = "01011001") then  -- 'Y'
    nxt_state <= s3;
  ELSif (regex_in = "01011010") then  -- 'Z'
    nxt_state <= s3;
  ELSif (regex_in = "01011011") then  -- '['
    nxt_state <= s3;
  ELSif (regex_in = "01011100") then  -- '\'
    nxt_state <= s3;
  ELSif (regex_in = "01011101") then  -- ']'
    nxt_state <= s3;
  ELSif (regex_in = "01011110") then  -- '^'
    nxt_state <= s3;
  ELSif (regex_in = "01011111") then  -- '_'
    nxt_state <= s3;
  ELSif (regex_in = "01100000") then  -- '`'
    nxt_state <= s3;
```

```
ELSif (regex_in = "01100001") then  -- 'a'
  nxt_state <= s3;
ELSif (regex_in = "01100010") then  -- 'b'
  nxt_state <= s3;
ELSif (regex_in = "01100011") then  -- 'c'
  nxt_state <= s3;
ELSif (regex_in = "01100100") then  -- 'd'
  nxt_state <= s3;
ELSif (regex_in = "01100101") then  -- 'e'
  nxt_state <= s3;
ELSif (regex_in = "01100110") then  -- 'f'
  nxt_state <= s3;
ELSif (regex_in = "01100111") then  -- 'g'
  nxt_state <= s3;
ELSif (regex_in = "01101000") then  -- 'h'
  nxt_state <= s3;
ELSif (regex_in = "01101001") then  -- 'i'
  nxt_state <= s3;
ELSif (regex_in = "01101010") then  -- 'j'
  nxt_state <= s3;
ELSif (regex_in = "01101011") then  -- 'k'
  nxt_state <= s3;
ELSif (regex_in = "01101100") then  -- 'l'
  nxt_state <= s3;
ELSif (regex_in = "01101101") then  -- 'm'
  nxt_state <= s3;
ELSif (regex_in = "01101110") then  -- 'n'
  nxt_state <= s3;
ELSif (regex_in = "01101111") then  -- 'o'
  nxt_state <= s3;
ELSif (regex_in = "01110000") then  -- 'p'
  nxt_state <= s3;
ELSif (regex_in = "01110001") then  -- 'q'
  nxt_state <= s3;
ELSif (regex_in = "01110010") then  -- 'r'
  nxt_state <= s3;
ELSif (regex_in = "01110011") then  -- 's'
  nxt_state <= s3;
ELSif (regex_in = "01110100") then  -- 't'
  nxt_state <= s2;
ELSif (regex_in = "01110101") then  -- 'u'
  nxt_state <= s3;
ELSif (regex_in = "01110110") then  -- 'v'
  nxt_state <= s3;
ELSif (regex_in = "01110111") then  -- 'w'
  nxt_state <= s3;
ELSif (regex_in = "01111000") then  -- 'x'
  nxt_state <= s3;
ELSif (regex_in = "01111001") then  -- 'y'
  nxt_state <= s3;
ELSif (regex_in = "01111010") then  -- 'z'
  nxt_state <= s3;
ELSif (regex_in = "01111011") then  -- '{'
  nxt_state <= s3;
ELSif (regex_in = "01111100") then  -- '|'
  nxt_state <= s3;
ELSif (regex_in = "01111101") then  -- '}'
  nxt_state <= s3;
ELSif (regex_in = "01111110") then  -- '~'
  nxt_state <= s3;
ELSif (regex_in = "01111111") then  -- DEL
```

```
                nxt_state <= s3;
              else
                nxt_state <= s0;
              end if;

      when s3 => if (regex_in = "00000001") then  -- SOH
                nxt_state <= s3;
              ELSif (regex_in = "00000010") then  -- STX
                nxt_state <= s3;
              ELSif (regex_in = "00000011") then  -- ETX
                nxt_state <= s3;
              ELSif (regex_in = "00000100") then  -- EOT
                nxt_state <= s3;
              ELSif (regex_in = "00000101") then  -- ENQ
                nxt_state <= s3;
              ELSif (regex_in = "00000110") then  -- ACK
                nxt_state <= s3;
              ELSif (regex_in = "00000111") then  -- BEL
                nxt_state <= s3;
              ELSif (regex_in = "00001000") then  -- BS
                nxt_state <= s3;
              ELSif (regex_in = "00001001") then  -- HT
                nxt_state <= s3;
              ELSif (regex_in = "00001011") then  -- VT
                nxt_state <= s3;
              ELSif (regex_in = "00001100") then  -- FF
                nxt_state <= s3;
              ELSif (regex_in = "00001110") then  -- SO
                nxt_state <= s3;
              ELSif (regex_in = "00001111") then  -- SI
                nxt_state <= s3;
              ELSif (regex_in = "00010000") then  -- DLE
                nxt_state <= s3;
              ELSif (regex_in = "00010001") then  -- DC1
                nxt_state <= s3;
              ELSif (regex_in = "00010010") then  -- DC2
                nxt_state <= s3;
              ELSif (regex_in = "00010011") then  -- DC3
                nxt_state <= s3;
              ELSif (regex_in = "00010100") then  -- DC4
                nxt_state <= s3;
              ELSif (regex_in = "00010101") then  -- NAK
                nxt_state <= s3;
              ELSif (regex_in = "00010110") then  -- SYN
                nxt_state <= s3;
              ELSif (regex_in = "00010111") then  -- ETB
                nxt_state <= s3;
              ELSif (regex_in = "00011000") then  -- CAN
                nxt_state <= s3;
              ELSif (regex_in = "00011001") then  -- EM
                nxt_state <= s3;
              ELSif (regex_in = "00011010") then  -- SUB
                nxt_state <= s3;
              ELSif (regex_in = "00011011") then  -- ESC
                nxt_state <= s3;
              ELSif (regex_in = "00011100") then  -- FSP
                nxt_state <= s3;
              ELSif (regex_in = "00011101") then  -- GSP
                nxt_state <= s3;
              ELSif (regex_in = "00011110") then  -- RSP
                nxt_state <= s3;
```

```
ELSif (regex_in = "00011111") then  -- USP
  nxt_state <= s3;
ELSif (regex_in = "00100000") then  -- ' '
  nxt_state <= s3;
ELSif (regex_in = "00100001") then  -- '!'
  nxt_state <= s3;
ELSif (regex_in = "00100010") then  -- '"'
  nxt_state <= s3;
ELSif (regex_in = "00100011") then  -- '#'
  nxt_state <= s3;
ELSif (regex_in = "00100100") then  -- '$'
  nxt_state <= s3;
ELSif (regex_in = "00100101") then  -- '%'
  nxt_state <= s3;
ELSif (regex_in = "00100110") then  -- '&'
  nxt_state <= s3;
ELSif (regex_in = "00100111") then  -- '''
  nxt_state <= s3;
ELSif (regex_in = "00101000") then  -- '('
  nxt_state <= s3;
ELSif (regex_in = "00101001") then  -- ')'
  nxt_state <= s3;
ELSif (regex_in = "00101010") then  -- '*'
  nxt_state <= s3;
ELSif (regex_in = "00101011") then  -- '+'
  nxt_state <= s3;
ELSif (regex_in = "00101100") then  -- ','
  nxt_state <= s3;
ELSif (regex_in = "00101101") then  -- '-'
  nxt_state <= s3;
ELSif (regex_in = "00101110") then  -- '.'
  nxt_state <= s3;
ELSif (regex_in = "00101111") then  -- '/'
  nxt_state <= s3;
ELSif (regex_in = "00110000") then  -- '0'
  nxt_state <= s3;
ELSif (regex_in = "00110001") then  -- '1'
  nxt_state <= s3;
ELSif (regex_in = "00110010") then  -- '2'
  nxt_state <= s3;
ELSif (regex_in = "00110011") then  -- '3'
  nxt_state <= s3;
ELSif (regex_in = "00110100") then  -- '4'
  nxt_state <= s3;
ELSif (regex_in = "00110101") then  -- '5'
  nxt_state <= s3;
ELSif (regex_in = "00110110") then  -- '6'
  nxt_state <= s3;
ELSif (regex_in = "00110111") then  -- '7'
  nxt_state <= s3;
ELSif (regex_in = "00111000") then  -- '8'
  nxt_state <= s3;
ELSif (regex_in = "00111001") then  -- '9'
  nxt_state <= s3;
ELSif (regex_in = "00111010") then  -- ':'
  nxt_state <= s3;
ELSif (regex_in = "00111011") then  -- ';'
  nxt_state <= s3;
ELSif (regex_in = "00111100") then  -- '<'
  nxt_state <= s3;
ELSif (regex_in = "00111101") then  -- '='
```

```
  nxt_state <= s3;
ELSif (regex_in = "00111110") then  -- '>'
  nxt_state <= s3;
ELSif (regex_in = "00111111") then  -- '?'
  nxt_state <= s3;
ELSif (regex_in = "01000000") then  -- '@'
  nxt_state <= s3;
ELSif (regex_in = "01000001") then  -- 'A'
  nxt_state <= s3;
ELSif (regex_in = "01000010") then  -- 'B'
  nxt_state <= s3;
ELSif (regex_in = "01000011") then  -- 'C'
  nxt_state <= s3;
ELSif (regex_in = "01000100") then  -- 'D'
  nxt_state <= s3;
ELSif (regex_in = "01000101") then  -- 'E'
  nxt_state <= s3;
ELSif (regex_in = "01000110") then  -- 'F'
  nxt_state <= s3;
ELSif (regex_in = "01000111") then  -- 'G'
  nxt_state <= s3;
ELSif (regex_in = "01001000") then  -- 'H'
  nxt_state <= s3;
ELSif (regex_in = "01001001") then  -- 'I'
  nxt_state <= s3;
ELSif (regex_in = "01001010") then  -- 'J'
  nxt_state <= s3;
ELSif (regex_in = "01001011") then  -- 'K'
  nxt_state <= s3;
ELSif (regex_in = "01001100") then  -- 'L'
  nxt_state <= s3;
ELSif (regex_in = "01001101") then  -- 'M'
  nxt_state <= s3;
ELSif (regex_in = "01001110") then  -- 'N'
  nxt_state <= s3;
ELSif (regex_in = "01001111") then  -- 'O'
  nxt_state <= s3;
ELSif (regex_in = "01010000") then  -- 'P'
  nxt_state <= s3;
ELSif (regex_in = "01010001") then  -- 'Q'
  nxt_state <= s3;
ELSif (regex_in = "01010010") then  -- 'R'
  nxt_state <= s3;
ELSif (regex_in = "01010011") then  -- 'S'
  nxt_state <= s3;
ELSif (regex_in = "01010100") then  -- 'T'
  nxt_state <= s3;
ELSif (regex_in = "01010101") then  -- 'U'
  nxt_state <= s3;
ELSif (regex_in = "01010110") then  -- 'V'
  nxt_state <= s3;
ELSif (regex_in = "01010111") then  -- 'W'
  nxt_state <= s3;
ELSif (regex_in = "01011000") then  -- 'X'
  nxt_state <= s3;
ELSif (regex_in = "01011001") then  -- 'Y'
  nxt_state <= s3;
ELSif (regex_in = "01011010") then  -- 'Z'
  nxt_state <= s3;
ELSif (regex_in = "01011011") then  -- '['
  nxt_state <= s3;
```

```
ELSif (regex_in = "01011100") then  -- '\'
  nxt_state <= s3;
ELSif (regex_in = "01011101") then  -- ']'
  nxt_state <= s3;
ELSif (regex_in = "01011110") then  -- '^'
  nxt_state <= s3;
ELSif (regex_in = "01011111") then  -- '_'
  nxt_state <= s3;
ELSif (regex_in = "01100000") then  -- '`'
  nxt_state <= s3;
ELSif (regex_in = "01100001") then  -- 'a'
  nxt_state <= s3;
ELSif (regex_in = "01100010") then  -- 'b'
  nxt_state <= s3;
ELSif (regex_in = "01100011") then  -- 'c'
  nxt_state <= s3;
ELSif (regex_in = "01100100") then  -- 'd'
  nxt_state <= s3;
ELSif (regex_in = "01100101") then  -- 'e'
  nxt_state <= s3;
ELSif (regex_in = "01100110") then  -- 'f'
  nxt_state <= s3;
ELSif (regex_in = "01100111") then  -- 'g'
  nxt_state <= s3;
ELSif (regex_in = "01101000") then  -- 'h'
  nxt_state <= s3;
ELSif (regex_in = "01101001") then  -- 'i'
  nxt_state <= s3;
ELSif (regex_in = "01101010") then  -- 'j'
  nxt_state <= s3;
ELSif (regex_in = "01101011") then  -- 'k'
  nxt_state <= s3;
ELSif (regex_in = "01101100") then  -- 'l'
  nxt_state <= s3;
ELSif (regex_in = "01101101") then  -- 'm'
  nxt_state <= s3;
ELSif (regex_in = "01101110") then  -- 'n'
  nxt_state <= s3;
ELSif (regex_in = "01101111") then  -- 'o'
  nxt_state <= s3;
ELSif (regex_in = "01110000") then  -- 'p'
  nxt_state <= s3;
ELSif (regex_in = "01110001") then  -- 'q'
  nxt_state <= s3;
ELSif (regex_in = "01110010") then  -- 'r'
  nxt_state <= s3;
ELSif (regex_in = "01110011") then  -- 's'
  nxt_state <= s3;
ELSif (regex_in = "01110100") then  -- 't'
  nxt_state <= s2;
ELSif (regex_in = "01110101") then  -- 'u'
  nxt_state <= s3;
ELSif (regex_in = "01110110") then  -- 'v'
  nxt_state <= s3;
ELSif (regex_in = "01110111") then  -- 'w'
  nxt_state <= s3;
ELSif (regex_in = "01111000") then  -- 'x'
  nxt_state <= s3;
ELSif (regex_in = "01111001") then  -- 'y'
  nxt_state <= s3;
ELSif (regex_in = "01111010") then  -- 'z'
```

```
                        nxt_state <= s3;
                      ELSif (regex_in = "01111011") then  -- '{'
                        nxt_state <= s3;
                      ELSif (regex_in = "01111100") then  -- '|'
                        nxt_state <= s3;
                      ELSif (regex_in = "01111101") then  -- '}'
                        nxt_state <= s3;
                      ELSif (regex_in = "01111110") then  -- '~'
                        nxt_state <= s3;
                      ELSif (regex_in = "01111111") then  -- DEL
                        nxt_state <= s3;
                      else
                        nxt_state <= s0;
                      end if;

      end case;
    end if;
  end process state_trans;
---------------------------
-- CONCURRENT STATEMENTS --
---------------------------

  accepting <= '1' when nxt_state = s1 or nxt_state = s2 else '0';
  reseting  <= '1' when (state /= s0) and (nxt_state  = s0) else '0';
  running   <= '1' when state /= s0 else '0';

end regex_arch1;
```

rad_loopback_core.vhd

```
--  applied research laboratory
--  washington university in st. louis
--
--  file:  rad_loopback_core.vhd
--  top level structure for rad fpga with ingress/egress loopback
modules
--  created by:  john w. lockwood (lockwood@arl.wustl.edu),
--  david e. taylor (det3@arl.wustl.edu)
--
library ieee;
use ieee.std_logic_1164.all;

entity rad_loopback_core is
  port (
    -- clocks
    rad_clk  : in std_logic;
    rad_clkb : in std_logic;

    -- reset & reconfig
    rad_reset : in std_logic;
    rad_ready : out std_logic;

    -- ingress path
    -- input
    soc_lc_nid   : in  std_logic;
    d_lc_nid     : in  std_logic_vector(31 downto 0);
    tcaff_lc_rad : out std_logic;
    -- output
    soc_lc_rad   : out std_logic;
    d_lc_rad     : out std_logic_vector(31 downto 0);
    tcaff_lc_nid : in  std_logic;

    -- egress path
    -- input
    soc_sw_nid   : in  std_logic;
    d_sw_nid     : in  std_logic_vector(31 downto 0);
    tcaff_sw_rad : out std_logic;
    -- output
    soc_sw_rad   : out std_logic;
    d_sw_rad     : out std_logic_vector(31 downto 0);
    tcaff_sw_nid : in  std_logic;

    -- test connector pins
    rad_test1 : out std_logic_vector(15 downto 0);
    rad_test2 : out std_logic_vector(15 downto 0);

    -- test led pins
    rad_led1 : out std_logic;
    rad_led2 : out std_logic;
    rad_led3 : out std_logic;
    rad_led4 : out std_logic
    );
end rad_loopback_core;
------------------------------------------------------------------------
-------
architecture structure of rad_loopback_core is

-- component declarations

  component loopback_module
    port (
```

```
      clk          : in  std_logic;
      reset_l      : in  std_logic;
      soc_mod_in   : in  std_logic;
      d_mod_in     : in  std_logic_vector(31 downto 0);
      tca_mod_in   : out std_logic;
      soc_out_mod  : out std_logic;
      d_out_mod    : out std_logic_vector(31 downto 0);
      tca_out_mod  : in  std_logic;
      test_data    : out std_logic_vector(31 downto 0)
      );
  end component;

  component regex_module
    port (
      clk          : in  std_logic;
      reset_l      : in  std_logic;
      soc_mod_in   : in  std_logic;
      d_mod_in     : in  std_logic_vector(31 downto 0);
      tca_mod_in   : out std_logic;
      soc_out_mod  : out std_logic;
      d_out_mod    : out std_logic_vector(31 downto 0);
      tca_out_mod  : in  std_logic;
      enable_l     : in  std_logic;
      ready_l      : out std_logic;
      test_data    : out std_logic_vector(31 downto 0)
      );
  end component;

  component blink
    port (
      clk1    : in  std_logic;
      clk2    : in  std_logic;
      reset_l : in  std_logic;
      led1    : out std_logic;
      led2    : out std_logic);
  end component;

-- signal declarations

  signal ingress_test, egress_test : std_logic_vector(31 downto 0);
  signal logic0, logic1 : std_logic;

begin  -- structural

  rad_ready <= not(rad_reset);

-- test pin flops
  test_pin_ff : process (rad_clk)
  begin  -- process test_pin_ff
    if rad_clk'event and rad_clk = '1' then  -- rising clock edge
        rad_test2 <= ingress_test(31 downto 16);
        rad_test1 <= ingress_test(15 downto 0);

        rad_led3 <= rad_reset;
        rad_led4 <= not rad_reset;
    end if;
  end process test_pin_ff;

  logic0 <= '0';
  logic1 <= '1';
```

```
ingress : regex_module
  port map (
    clk          => rad_clk,
    reset_l      => rad_reset,
    soc_mod_in   => soc_lc_nid,
    d_mod_in     => d_lc_nid,
    tca_mod_in   => tcaff_lc_rad,
    soc_out_mod  => soc_lc_rad,
    d_out_mod    => d_lc_rad,
    tca_out_mod  => tcaff_lc_nid,
    enable_l     => logic0,
    ready_l      => open,
    test_data    => ingress_test);

egress : loopback_module
  port map (
    clk          => rad_clkb,
    reset_l      => rad_reset,
    soc_mod_in   => soc_sw_nid,
    d_mod_in     => d_sw_nid,
    tca_mod_in   => tcaff_sw_rad,
    soc_out_mod  => soc_sw_rad,
    d_out_mod    => d_sw_rad,
    tca_out_mod  => tcaff_sw_nid,
    test_data    => egress_test);

blink1 : blink
  port map (
    clk1    => rad_clk,
    clk2    => rad_clkb,
    reset_l => rad_reset,
    led1    => rad_led1,
    led2    => rad_led2);

end structure;
```

rad_loopback.vhd

```
--  applied research laboratory
--  washington university in st. louis
--
--  file:  rad_loopback.vhd
--  top level structure for rad fpga with ingress/egress loopback
modules
--  created by:  john w. lockwood (lockwood@arl.wustl.edu),
--  david e. taylor (det3@arl.wustl.edu)
--
------------------------------------------------------------------------
-------
library ieee;
use ieee.std_logic_1164.all;
-- synthesis translate_off
library unisim;
-- synthesis translate_on
------------------------------------------------------------------------
-------
entity rad_loopback is
  port (
    -- clocks
    rad_clk  : in std_logic;
    rad_clkb : in std_logic;

    -- reset & reconfig
    rad_reset : in  std_logic;
    rad_ready : out std_logic;
    rad_reconfig : inout std_logic_vector(2 downto 0);

    -- nid interface
    --
    -- ingress path
    -- input
    soc_lc_nid    : in  std_logic;
    d_lc_nid      : in  std_logic_vector(31 downto 0);
    tcaff_lc_rad  : out std_logic;
    -- output
    soc_lc_rad    : out std_logic;
    d_lc_rad      : out std_logic_vector(31 downto 0);
    tcaff_lc_nid  : in  std_logic;

    -- egress path
    -- input
    soc_sw_nid    : in  std_logic;
    d_sw_nid      : in  std_logic_vector(31 downto 0);
    tcaff_sw_rad  : out std_logic;
    -- output
    soc_sw_rad    : out std_logic;
    d_sw_rad      : out std_logic_vector(31 downto 0);
    tcaff_sw_nid  : in  std_logic;

    -- test connector pins
    rad_test1 : out std_logic_vector(15 downto 0);
    rad_test2 : out std_logic_vector(15 downto 0);

    -- test led pins
    rad_led1 : out std_logic;
    rad_led2 : out std_logic;
    rad_led3 : out std_logic;
    rad_led4 : out std_logic
    );
```

```
end rad_loopback;
--------------------------------------------------------------------------
-------
architecture structure of rad_loopback is

-- component declarations

  component iobuf_f_12
    port (
      o  : out   std_ulogic;
      i  : in    std_ulogic;
      io : inout std_logic;
      t  : in    std_logic
      );
  end component;

  component bufgdll
    port ( o : out std_ulogic;
           i : in  std_ulogic
           );
  end component;

-- synthesis translate_off
  for all : iobuf_f_12 use entity unisim.iobuf_f_12(iobuf_f_12_v);
  for all : bufgdll use entity unisim.bufgdll(bufgdll_v);
-- synthesis translate_on

  component rad_loopback_core
    port (
      rad_clk       : in    std_logic;
      rad_clkb      : in    std_logic;
      rad_reset     : in    std_logic;
      rad_ready     : out   std_logic;
      soc_lc_nid    : in    std_logic;
      d_lc_nid      : in    std_logic_vector(31 downto 0);
      tcaff_lc_rad  : out   std_logic;
      soc_lc_rad    : out   std_logic;
      d_lc_rad      : out   std_logic_vector(31 downto 0);
      tcaff_lc_nid  : in    std_logic;
      soc_sw_nid    : in    std_logic;
      d_sw_nid      : in    std_logic_vector(31 downto 0);
      tcaff_sw_rad  : out   std_logic;
      soc_sw_rad    : out   std_logic;
      d_sw_rad      : out   std_logic_vector(31 downto 0);
      tcaff_sw_nid  : in    std_logic;
      rad_test1     : out std_logic_vector(15 downto 0);
      rad_test2     : out std_logic_vector(15 downto 0);
      rad_led1      : out   std_logic;
      rad_led2      : out   std_logic;
      rad_led3      : out   std_logic;
      rad_led4      : out   std_logic);
  end component;
--------------------------------------------------------------------------
-------

-- signal declarations

  signal rad_clk_dll      : std_logic;  -- dll clock output
  signal rad_clkb_dll     : std_logic;  -- dll clock output

  signal rad_reset_i      : std_logic;
```

```
 signal rad_ready_i       : std_logic;
 signal soc_lc_nid_i      : std_logic;
 signal d_lc_nid_i        : std_logic_vector(31 downto 0);
 signal tcaff_lc_rad_i    : std_logic;
 signal soc_lc_rad_i      : std_logic;
 signal d_lc_rad_i        : std_logic_vector(31 downto 0);
 signal tcaff_lc_nid_i    : std_logic;
 signal soc_sw_nid_i      : std_logic;
 signal d_sw_nid_i        : std_logic_vector(31 downto 0);
 signal tcaff_sw_rad_i    : std_logic;
 signal soc_sw_rad_i      : std_logic;
 signal d_sw_rad_i        : std_logic_vector(31 downto 0);
 signal tcaff_sw_nid_i    : std_logic;
 signal rad_test1_i       : std_logic_vector(15 downto 0);
 signal rad_test2_i       : std_logic_vector(15 downto 0);
 signal rad_led1_i        : std_logic;
 signal rad_led2_i        : std_logic;
 signal rad_led3_i        : std_logic;
 signal rad_led4_i        : std_logic;

 signal rad_reset_pad     : std_logic;
 signal rad_ready_pad     : std_logic;
 signal soc_lc_nid_pad    : std_logic;
 signal d_lc_nid_pad      : std_logic_vector(31 downto 0);
 signal tcaff_lc_rad_pad  : std_logic;
 signal soc_lc_rad_pad    : std_logic;
 signal d_lc_rad_pad      : std_logic_vector(31 downto 0);
 signal tcaff_lc_nid_pad  : std_logic;
 signal soc_sw_nid_pad    : std_logic;
 signal d_sw_nid_pad      : std_logic_vector(31 downto 0);
 signal tcaff_sw_rad_pad  : std_logic;
 signal soc_sw_rad_pad    : std_logic;
 signal d_sw_rad_pad      : std_logic_vector(31 downto 0);
 signal tcaff_sw_nid_pad  : std_logic;
 signal rad_test1_pad     : std_logic_vector(15 downto 0);
 signal rad_test2_pad     : std_logic_vector(15 downto 0);
 signal rad_led1_pad      : std_logic;
 signal rad_led2_pad      : std_logic;
 signal rad_led3_pad      : std_logic;
 signal rad_led4_pad      : std_logic;
-------------------------------------------------------------------------
-------
begin  -- structural

  rad_clk_dll  <= rad_clk;
  rad_clkb_dll <= rad_clkb;

  -- purpose: double buffer all off-chip signals
  iob_flops : process (rad_clk_dll)
  begin  -- process iob_flops
    if rad_clk_dll'event and rad_clk_dll = '1' then  -- rising clock
edge
      rad_reset_pad    <= rad_reset;
      rad_ready        <= rad_ready_pad;
      soc_lc_nid_pad   <= soc_lc_nid;
      d_lc_nid_pad     <= d_lc_nid;
      tcaff_lc_rad     <= tcaff_lc_rad_pad;
      soc_lc_rad       <= soc_lc_rad_pad;
      d_lc_rad         <= d_lc_rad_pad;
      tcaff_lc_nid_pad <= tcaff_lc_nid;
      soc_sw_nid_pad   <= soc_sw_nid;
```

```
      d_sw_nid_pad      <= d_sw_nid;
      tcaff_sw_rad      <= tcaff_sw_rad_pad;
      soc_sw_rad        <= soc_sw_rad_pad;
      d_sw_rad          <= d_sw_rad_pad;
      tcaff_sw_nid_pad  <= tcaff_sw_nid;
      rad_test1         <= rad_test1_pad;
      rad_test2         <= rad_test2_pad;
      rad_led1          <= rad_led1_pad;
      rad_led2          <= rad_led2_pad;
      rad_led3          <= rad_led3_pad;
      rad_led4          <= rad_led4_pad;

      rad_reset_i       <= rad_reset_pad;
      rad_ready_pad     <= rad_ready_i;
      soc_lc_nid_i      <= soc_lc_nid_pad;
      d_lc_nid_i        <= d_lc_nid_pad;
      tcaff_lc_rad_pad  <= tcaff_lc_rad_i;
      soc_lc_rad_pad    <= soc_lc_rad_i;
      d_lc_rad_pad      <= d_lc_rad_i;
      tcaff_lc_nid_i    <= tcaff_lc_nid_pad;
      soc_sw_nid_i      <= soc_sw_nid_pad;
      d_sw_nid_i        <= d_sw_nid_pad;
      tcaff_sw_rad_pad  <= tcaff_sw_rad_i;
      soc_sw_rad_pad    <= soc_sw_rad_i;
      d_sw_rad_pad      <= d_sw_rad_i;
      tcaff_sw_nid_i    <= tcaff_sw_nid_pad;
      rad_test1_pad     <= rad_test1_i;
      rad_test2_pad     <= rad_test2_i;
      rad_led1_pad      <= rad_led1_i;
      rad_led2_pad      <= rad_led2_i;
      rad_led3_pad      <= rad_led3_i;
      rad_led4_pad      <= rad_led4_i;
    end if;
  end process iob_flops;

  rad_loopback_core_1 : rad_loopback_core
    port map (
      rad_clk       => rad_clk_dll,
      rad_clkb      => rad_clkb_dll,
      rad_reset     => rad_reset_i,
      rad_ready     => rad_ready_i,
      soc_lc_nid    => soc_lc_nid_i,
      d_lc_nid      => d_lc_nid_i,
      tcaff_lc_rad  => tcaff_lc_rad_i,
      soc_lc_rad    => soc_lc_rad_i,
      d_lc_rad      => d_lc_rad_i,
      tcaff_lc_nid  => tcaff_lc_nid_i,
      soc_sw_nid    => soc_sw_nid_i,
      d_sw_nid      => d_sw_nid_i,
      tcaff_sw_rad  => tcaff_sw_rad_i,
      soc_sw_rad    => soc_sw_rad_i,
      d_sw_rad      => d_sw_rad_i,
      tcaff_sw_nid  => tcaff_sw_nid_i,
      rad_test1     => rad_test1_i,
      rad_test2     => rad_test2_i,
      rad_led1      => rad_led1_i,
      rad_led2      => rad_led2_i,
      rad_led3      => rad_led3_i,
      rad_led4      => rad_led4_i);
end structure;
```

loopback_module.vhd

```
--  applied research laboratory
--  washington university in st. louis
--
--  file: loopback_module.vhd
--  entity declaration for rad module
--  created by:  david e. taylor (det3@arl.wustl.edu)
--  created on:  august, 16 2000
--  last modified:  august 18, 2000 @ 11:20 am
--
--  important:  refer to rad module interface specification
--  for explanations and timing specifications for all interface
--  signals.

library ieee;
use ieee.std_logic_1164.all;

entity loopback_module is
  port (
    -- clock & reset
    clk     : in std_logic;              -- 100mhz global clock
    reset_l : in std_logic;              -- synchronous reset, asserted-
low

    -- cell input interface
    soc_mod_in    : in  std_logic;                    -- start of cell
    d_mod_in      : in  std_logic_vector(31 downto 0);  -- 32-bit data
    tca_mod_in    : out std_logic;                    -- transmit cell
available

    -- cell output interface
    soc_out_mod   : out std_logic;                    -- start of cell
    d_out_mod     : out std_logic_vector(31 downto 0);  -- 32-bit data
    tca_out_mod   : in  std_logic;                     -- transmit
cell available
    test_data     : out std_logic_vector(31 downto 0)   -- 32-bit data
    );
end loopback_module;

architecture behavioral of loopback_module is

  signal soc : std_logic;                 -- start of cell
  signal data : std_logic_vector(31 downto 0);  -- 32-bit words of atm
cell
  signal tca : std_logic;                 -- transmit cell available

begin  -- behavioral

-- pass cells through.  on reconfig, hold tca low.

  cell_io_ff : process (clk)
  begin  -- process cell_ff
    if (clk='1' and clk'event) then
      if reset_l='0' then
        soc <= '0';
        soc_out_mod <= '0';
        data <= (others=>'0') ;
        d_out_mod <= (others=>'0') ;
        tca <= '0' ;
        tca_mod_in <= '0' ;
      else
        soc <= soc_mod_in ;
```

```
        data <= d_mod_in ;
        soc_out_mod <= soc ;
        d_out_mod <= data ;
        tca <= tca_out_mod ;
        tca_mod_in <= tca ;
      end if;
    end if;
  end process cell_io_ff;

  test_data <= data;

end behavioral;
```

blink.vhd

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_unsigned.all;

entity blink is
  port (
    clk1     : in  std_logic;
    clk2     : in  std_logic;
    reset_l  : in  std_logic;
    led1     : out std_logic;
    led2     : out std_logic
  );
end blink;

architecture behav of blink is

signal led1_27     : std_logic_vector(26 downto 0);
signal led2_27     : std_logic_vector(26 downto 0);

begin

-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * *
-- led1 is driven by a 27-bit counter clocked by the rad system clock
(100mhz)
-- led2 is driven by a 27-bit counter clocked by the rad system clock b
(100mhz)
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * *

-- a flip-flop is added after the last counter stage so that it can
-- sit in the iob.

-- led1 logic
led1_proc : process (clk1, reset_l)
  begin
    if (reset_l = '0') then
      led1 <=  '1';
    elsif (clk1'event and clk1 = '1') then
      led1 <= led1_27(26);
    end if;
  end process;



-- divide by 67 million
led1_cntr_proc : process (clk1, reset_l)
  begin
    if (reset_l = '0') then
      led1_27 <= (others => '0');
    elsif (clk1'event and clk1 = '1') then
      led1_27 <= unsigned (led1_27) + 1;
    end if;
  end process;

-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * *
-- led2 logic
led2_proc : process (clk2, reset_l)
  begin
```

```
    if (reset_l = '0') then
      led2 <=  '1';
    elsif (clk2'event and clk2 = '1') then
      led2 <= led2_27(26);
    end if;
  end process;


-- divide by 67 million
led2_cntr_proc : process (clk2, reset_l)
  begin
    if (reset_l = '0') then
      led2_27 <= (others => '0');
    elsif (clk2'event and clk2 = '1') then
      led2_27 <= unsigned (led2_27) + 1;
    end if;
  end process;

end behav;
```

regex_module.vhd

```
library ieee;
use ieee.std_logic_1164.all;


entity regex_module is
  port (
    -- clock & reset
    clk     : in std_logic;                -- 100mhz global clock
    reset_l : in std_logic;                -- synchronous reset, asserted-
low

    -- enable & ready
    -- handshake for module reconfiguration.
    enable_l : in  std_logic;              -- asserted low
    ready_l  : out std_logic;              -- asserted low

    -- cell input interface
    soc_mod_in   : in  std_logic;                        -- start of cell
    d_mod_in     : in  std_logic_vector(31 downto 0);  -- 32-bit data
    tca_mod_in   : out std_logic;                        -- transmit cell
 available

    -- cell output interface
    soc_out_mod   : out std_logic;                        -- start of cell
    d_out_mod     : out std_logic_vector(31 downto 0);  -- 32-bit data
    tca_out_mod   : in  std_logic;

    -- test data output
    test_data     : out std_logic_vector(31 downto 0));

 end regex_module;

 architecture struc of regex_module is

   component udpwrapper
     port (
       clk              : in  std_logic;    -- clock
       reset_l          : in  std_logic;    -- reset
       enable_l         : in  std_logic;    -- enable
       ready_l          : out std_logic;    -- ready
       soc_mod_in       : in  std_logic;    -- start of cell
       d_mod_in         : in  std_logic_vector (31 downto 0);  -- data
       tca_mod_in       : out std_logic;    -- transmit cell available
       d_out_appl       : out std_logic_vector (31 downto 0);  -- data to
 appl
       dataen_out_appl  : out std_logic;    -- data enable
       sof_out_appl     : out std_logic;    -- start of frame
       eof_out_appl     : out std_logic;    -- end of frame
       sod_out_appl     : out std_logic;    -- start of datagram
       tca_out_appl     : in  std_logic;    -- congestion control
       d_appl_in        : in  std_logic_vector (31 downto 0);  -- data
 from appl
       dataen_appl_in   : in  std_logic;    -- data enable
       sof_appl_in      : in  std_logic;    -- start of frame
       eof_appl_in      : in  std_logic;    -- end of frame
       sod_appl_in      : in  std_logic;    -- start of datagram
       tca_appl_in      : out std_logic;    -- congestion control
       soc_out_mod      : out std_logic;    -- start of cell
       d_out_mod        : out std_logic_vector (31 downto 0);  -- data
       tca_out_mod      : in  std_logic;   -- transmit cell available
```

```
    jmos_soc_out_cell  : out std_logic;
    jmos_soc_out_frame : out std_logic;
    jmos_soc_out_ip : out std_logic;
    jmos_soc_out_udp : out std_logic);
  end component;

-------------------------------------------------------------------------
-----
  -- interface
-------------------------------------------------------------------------
-----

  component regex_app
    port (
      clk               : in  std_logic;                         -- clock
      reset_l           : in  std_logic;                         -- reset
      enable_l          : in std_logic;
      ready_l           : out std_logic;
      dataen_out_appl   : in  std_logic;                         -- data
enable
      d_out_appl        : in  std_logic_vector (31 downto 0); -- data
      sof_out_appl      : in  std_logic;                         -- start of
frame
      eof_out_appl      : in  std_logic;                         -- end of
frame
      sod_out_appl      : in  std_logic;                         -- start of
datagram
      tca_appl_in       : in  std_logic;                         --
congestion control
      dataen_appl_in    : out std_logic;                         -- data
enable
      d_appl_in         : out std_logic_vector (31 downto 0); -- data
      sof_appl_in       : out std_logic;                         -- start of
frame
      eof_appl_in       : out std_logic;                         -- end of
frame
      sod_appl_in       : out std_logic;                         -- start of
datagram
      tca_out_appl      : out  std_logic);                       --
congestion control
  end component;

-------------------------------------------------------------------------
-----
  -- signals udpproc - application
-------------------------------------------------------------------------
-----
  signal data_u2h : std_logic_vector (31 downto 0);   -- data
  signal dataen_u2h : std_logic;                      -- data enable
  signal sof_u2h : std_logic;                         -- start of frame
  signal eof_u2h : std_logic;                         -- end of frame
  signal sod_u2h : std_logic;                         -- start of payload
  signal tca_u2h : std_logic;                         -- congestion
control

-------------------------------------------------------------------------
```

```
-----
  -- signals application - udpproc
--------------------------------------------------------------------------
-----
  signal data_h2u : std_logic_vector (31 downto 0);  -- data
  signal dataen_h2u : std_logic;                     -- data enable
  signal sof_h2u : std_logic;                        -- start of frame
  signal eof_h2u : std_logic;                        -- end of frame
  signal sod_h2u : std_logic;                        -- start of payload
  signal tca_h2u : std_logic;                        -- congestion
control
  signal ready1_l : std_logic;
  signal ready2_l : std_logic;

  signal jmos_soc_out_cell : std_logic;
  signal jmos_soc_out_frame : std_logic;
  signal jmos_soc_out_ip   : std_logic;
  signal jmos_soc_out_udp : std_logic;
begin  -- struc

--------------------------------------------------------------------------
-----
  -- udpwrapper
--------------------------------------------------------------------------
-----
  upw : udpwrapper port map (
    clk           => clk,
    reset_l       => reset_l,
    enable_l      => enable_l,
    ready_l       => ready1_l,
    soc_mod_in    => soc_mod_in,
    d_mod_in      => d_mod_in,
    tca_mod_in    => tca_mod_in,
    soc_out_mod   => soc_out_mod,
    d_out_mod     => d_out_mod, -- open
    tca_out_mod   => tca_out_mod,
    d_out_appl       => data_u2h,
    dataen_out_appl  => dataen_u2h,
    sof_out_appl     => sof_u2h,
    eof_out_appl     => eof_u2h,
    sod_out_appl     => sod_u2h,
    tca_out_appl     => tca_u2h,
    d_appl_in        => data_h2u,
    dataen_appl_in   => dataen_h2u,
    sof_appl_in      => sof_h2u,
    eof_appl_in      => eof_h2u,
    sod_appl_in      => sod_h2u,
    tca_appl_in      => tca_h2u,

    jmos_soc_out_cell  => jmos_soc_out_cell,
    jmos_soc_out_frame => jmos_soc_out_frame,
    jmos_soc_out_ip => jmos_soc_out_ip,
    jmos_soc_out_udp => jmos_soc_out_udp);

--------------------------------------------------------------------------
-----
```

```
  -- regular expression application
---------------------------------------------------------------------------
-----
  app : regex_app port map (
    clk                  => clk,
    reset_l              => reset_l,
    enable_l             => enable_l,
    ready_l              => ready2_l,
    dataen_out_appl      => dataen_u2h,
    d_out_appl           => data_u2h,
    sof_out_appl         => sof_u2h,
    eof_out_appl         => eof_u2h,
    sod_out_appl         => sod_u2h,
    tca_appl_in          => tca_h2u,
    dataen_appl_in       => dataen_h2u,
    d_appl_in            => data_h2u,
    sof_appl_in          => sof_h2u,
    eof_appl_in          => eof_h2u,
    sod_appl_in          => sod_h2u,
    tca_out_appl         => tca_u2h);


---------------------------------------------------------------------------
-----
  -- test pins
---------------------------------------------------------------------------
-----

-- backside of my application
  test_data (31) <= sof_h2u;
  test_data (30) <= eof_h2u;
  test_data (29) <= sod_h2u;
  test_data (28) <= dataen_h2u;
  test_data (27 downto 16) <= data_h2u (11 downto 0);

  test_data (15) <= jmos_soc_out_cell;
  test_data (14) <= jmos_soc_out_frame;
  test_data (13) <= jmos_soc_out_ip;
  test_data (12) <= jmos_soc_out_udp;
  test_data (11 downto 0) <= "000000000000";

--  test_data (15) <= sof_u2h;
--  test_data (14) <= eof_u2h;
--  test_data (13) <= sod_u2h;
--  test_data (12) <= dataen_u2h;
--  test_data (11 downto 0) <= data_u2h (11 downto 0);

  ready_l <= ready1_l and ready2_l;

end struc;
```

```
--   applied research laboratory
--   washington university in st. louis
--
--   file:  rad_loopback_core.vhd
--   top level structure for rad fpga with ingress/egress loopback
modules
--   created by:  john w. lockwood (lockwood@arl.wustl.edu),
--   david e. taylor (det3@arl.wustl.edu)
--
library ieee;
use ieee.std_logic_1164.all;

entity rad_loopback_core is
  port (
    -- clocks
    rad_clk  : in std_logic;
    rad_clkb : in std_logic;

    -- reset & reconfig
    rad_reset : in std_logic;
    rad_ready : out std_logic;

    -- ingress path
    -- input
    soc_lc_nid   : in  std_logic;
    d_lc_nid     : in  std_logic_vector(31 downto 0);
    tcaff_lc_rad : out std_logic;
    -- output
    soc_lc_rad   : out std_logic;
    d_lc_rad     : out std_logic_vector(31 downto 0);
    tcaff_lc_nid : in  std_logic;

    -- egress path
    -- input
    soc_sw_nid   : in  std_logic;
    d_sw_nid     : in  std_logic_vector(31 downto 0);
    tcaff_sw_rad : out std_logic;
    -- output
    soc_sw_rad   : out std_logic;
    d_sw_rad     : out std_logic_vector(31 downto 0);
    tcaff_sw_nid : in  std_logic;

    -- test connector pins
    rad_test1 : out std_logic_vector(15 downto 0);
    rad_test2 : out std_logic_vector(15 downto 0);

    -- test led pins
    rad_led1 : out std_logic;
    rad_led2 : out std_logic;
    rad_led3 : out std_logic;
    rad_led4 : out std_logic
    );
 end rad_loopback_core;
 ------------------------------------------------------------------------
 -------
 architecture structure of rad_loopback_core is

 -- component declarations

   component loopback_module
     port (
```

```
      clk          : in  std_logic;
      reset_l      : in  std_logic;
      soc_mod_in   : in  std_logic;
      d_mod_in     : in  std_logic_vector(31 downto 0);
      tca_mod_in   : out std_logic;
      soc_out_mod  : out std_logic;
      d_out_mod    : out std_logic_vector(31 downto 0);
      tca_out_mod  : in  std_logic;
      test_data    : out std_logic_vector(31 downto 0)
      );
  end component;

  component regex_module
    port (
      clk          : in  std_logic;
      reset_l      : in  std_logic;
      soc_mod_in   : in  std_logic;
      d_mod_in     : in  std_logic_vector(31 downto 0);
      tca_mod_in   : out std_logic;
      soc_out_mod  : out std_logic;
      d_out_mod    : out std_logic_vector(31 downto 0);
      tca_out_mod  : in  std_logic;
      enable_l     : in  std_logic;
      ready_l      : out std_logic;
      test_data    : out std_logic_vector(31 downto 0)
      );
  end component;

  component blink
    port (
      clk1     : in  std_logic;
      clk2     : in  std_logic;
      reset_l  : in  std_logic;
      led1     : out std_logic;
      led2     : out std_logic);
  end component;

-- signal declarations

  signal ingress_test, egress_test : std_logic_vector(31 downto 0);
  signal logic0, logic1 : std_logic;

begin  -- structural

  rad_ready <= not(rad_reset);

-- test pin flops
  test_pin_ff : process (rad_clk)
  begin  -- process test_pin_ff
    if rad_clk'event and rad_clk = '1' then  -- rising clock edge
        rad_test2 <= ingress_test(31 downto 16);
        rad_test1 <= ingress_test(15 downto 0);

        rad_led3 <= rad_reset;
        rad_led4 <= not rad_reset;
    end if;
  end process test_pin_ff;

  logic0 <= '0';
  logic1 <= '1';
```

```
  ingress : regex_module
    port map (
      clk          => rad_clk,
      reset_l      => rad_reset,
      soc_mod_in   => soc_lc_nid,
      d_mod_in     => d_lc_nid,
      tca_mod_in   => tcaff_lc_rad,
      soc_out_mod => soc_lc_rad,
      d_out_mod    => d_lc_rad,
      tca_out_mod => tcaff_lc_nid,
      enable_l     => logic0,
      ready_l      => open,
      test_data    => ingress_test);

  egress : loopback_module
    port map (
      clk          => rad_clkb,
      reset_l      => rad_reset,
      soc_mod_in   => soc_sw_nid,
      d_mod_in     => d_sw_nid,
      tca_mod_in   => tcaff_sw_rad,
      soc_out_mod => soc_sw_rad,
      d_out_mod    => d_sw_rad,
      tca_out_mod => tcaff_sw_nid,
      test_data    => egress_test);

  blink1 : blink
    port map (
      clk1     => rad_clk,
      clk2     => rad_clkb,
      reset_l => rad_reset,
      led1     => rad_led1,
      led2     => rad_led2);

end structure;
```

MakeProject

```
#!/bin/gawk -f

BEGIN {

  outfile="regex_app.prj"
  if (n=="")
  {
    print "\nUSAGE: "
    print "  makeProject -v n={number}\n"
    print "    {number} :  The number of machines this hardware
contains"

  }else{
    print "add_file -vhdl -lib work \"../vhdl/wrappers/framewrapper.vhd
\"" > outfile
    print "add_file -vhdl -lib work \"../vhdl/wrappers/ipwrapper.vhd
\"" > outfile
    print "add_file -vhdl -lib work \"../vhdl/wrappers/udpwrapper.vhd
\"" > outfile
    for (i=1; i<=n; i++)
    {
      print "add_file -vhdl -lib work \"../vhdl/regex_fsm" i ".vhd\"" >
outfile
      print "add_file -vhdl -lib work \"../vhdl/replace_buf" i ".vhd
\"" > outfile
    }
    print "add_file -vhdl -lib work \"../vhdl/wrd_bldr.vhd\"" > outfile
    print "add_file -vhdl -lib work \"../vhdl/controller.vhd\"" >
outfile
    print "add_file -vhdl -lib work \"../vhdl/regex_app.vhd\"" > outfile
    print "add_file -vhdl -lib work
\"../vhdl/rad_loopback/regex_module.vhd\"" >outfile
    print "add_file -vhdl -lib work \"../vhdl/rad_loopback/blink.vhd
\"" >outfile
    print "add_file -vhdl -lib work
\"../vhdl/rad_loopback/loopback_module.vhd\"" >outfile
    print "add_file -vhdl -lib work
\"../vhdl/rad_loopback/rad_loopback_core.vhd\"" >outfile
    print "add_file -vhdl -lib work
\"../vhdl/rad_loopback/rad_loopback.vhd\"" >outfile
    print "" > outfile
    print "" > outfile
    print "impl -add regex_app" > outfile
    print "" > outfile
    print "set_option -technology VIRTEX-E" > outfile
    print "set_option -part XCV1000E" > outfile
    print "set_option -package FG680" > outfile
    print "set_option -speed_grade -7" > outfile
    print "" > outfile
    print "set_option -default_enum_encoding default" > outfile
    print "set_option -symbolic_fsm_compiler 1" > outfile
    print "set_option -resource_sharing 1" > outfile
    print "set_option -top_module \"rad_loopback\"" > outfile
    print "" > outfile
    print "set_option -frequency 100.000" > outfile
    print "set_option -fanout_limit 32" > outfile
    print "set_option -disable_io_insertion 0" > outfile
    print "set_option -pipe 0" > outfile
    print "set_option -modular 0" > outfile
    print "set_option -retiming 0" > outfile
    print "" > outfile
```

```
    print "set_option -write_verilog 0" > outfile
    print "set_option -write_vhdl 0" > outfile
    print "" > outfile
    print "set_option -write_apr_constraint 1" > outfile
    print "" > outfile
    print "project -result_format \"edif\"" > outfile
    print "project -result_file \"regex_app.edf\"" > outfile

  }
}
```

regex_app.prj

```
add_file -vhdl -lib work "../vhdl/wrappers/framewrapper.vhd"
add_file -vhdl -lib work "../vhdl/wrappers/ipwrapper.vhd"
add_file -vhdl -lib work "../vhdl/wrappers/udpwrapper.vhd"
add_file -vhdl -lib work "../vhdl/regex_fsm1.vhd"
add_file -vhdl -lib work "../vhdl/replace_buf1.vhd"
add_file -vhdl -lib work "../vhdl/wrd_bldr.vhd"
add_file -vhdl -lib work "../vhdl/controller.vhd"
add_file -vhdl -lib work "../vhdl/regex_app.vhd"
add_file -vhdl -lib work "../vhdl/rad_loopback/regex_module.vhd"
add_file -vhdl -lib work "../vhdl/rad_loopback/blink.vhd"
add_file -vhdl -lib work "../vhdl/rad_loopback/loopback_module.vhd"
add_file -vhdl -lib work "../vhdl/rad_loopback/rad_loopback_core.vhd"
add_file -vhdl -lib work "../vhdl/rad_loopback/rad_loopback.vhd"


impl -add regex_app

set_option -technology VIRTEX-E
set_option -part XCV1000E
set_option -package FG680
set_option -speed_grade -7

set_option -default_enum_encoding default
set_option -symbolic_fsm_compiler 1
set_option -resource_sharing 1
set_option -top_module "rad_loopback"

set_option -frequency 100.000
set_option -fanout_limit 32
set_option -disable_io_insertion 0
set_option -pipe 0
set_option -modular 0
set_option -retiming 0

set_option -write_verilog 0
set_option -write_vhdl 0

set_option -write_apr_constraint 1

project -result_format "edif"
project -result_file "regex_app.edf"
```

What Is Claimed Is:

1. A reprogrammable system for processing a stream of data, said system comprising:

a reprogrammable data processor for receiving a stream of data and processing said received data stream through a programmable logic device (PLD) programmed to (1) determine whether said data stream includes a string that matches a redefinable data pattern, and (2) perform a redefinable action in the event said data stream is found to include a string that matches said data pattern; and

a reconfiguration device in communication with said data processor that is operable to reprogram said PLD with at least one of the group consisting of a redefined data pattern and a redefined action.

2. The system of claim 1 wherein said redefinable action is a data modification operation, and wherein said PLD is programmed to perform said data modification operation by modifying at least a portion of a data stream that is found to include a matching string.

3. The system of claim 2 wherein said data modification operation is a string replacement operation, said string replacement operation including a redefinable replacement string, and wherein said PLD is programmed to perform said string replacement operation by replacing a matching string in said data stream with said replacement string.

4. The system of claim 3 wherein said PLD is programmed to perform said string replacement operation by replacing a longest matching string in said data stream with said replacement string.

5. The system of claim 3 wherein said data processor is in communication with a computer network from which said data stream is received, said data stream comprising a stream of data packets transmitted over said computer network.

6. The system of claim 5 wherein each packet in said packet stream includes a payload portion, and wherein said PLD is programmed to

determine whether the payload portion of any of said received packets includes a matching string.

7. The system of claim 6 wherein said reconfiguration device is in communication with said data processor via said computer network and is further operable to reprogram said PLD over said computer network.

8. The system of claim 6 wherein said reconfiguration device comprises:

a reconfiguration input operable to receive a data pattern and a replacement string;

a compiler operable to (1) generate a module from said received data pattern and said received replacement string that is operable upon being programmed into said PLD to determine whether the payload of a packet applied thereto includes a string that matches said received data pattern and, if a matching string is found therein, replace said matching string with said received replacement string, and (2) create configuration information from said generated module that is operable to program said PLD with said generated module; and

a transmitter operable to communicate said configuration information over said network to said data processor for programming said PLD with said module.

9. The system of claim 8 wherein said data processor further comprises a programming device in communication with said PLD and said computer network that is operable to receive said configuration information from said transmitter and program said PLD according to said received configuration information.

10. The system of claim 9 wherein said reconfiguration input is further configured to receive said data pattern in a regular expression format.

11. The system of claim 10 wherein said compiler is further configured to generate said module in part by processing said received data pattern through a lexical analyzer generator to thereby create a logical representation of a pattern matching state machine operable to determine whether data applied thereto includes a string

that matches said received data pattern, said module including said pattern matching state machine representation.

12. The system of claim 11 wherein said lexical analyzer generator is JLex.

13. The system of claim 11 wherein said module includes a plurality of said pattern matching state machines representations for parallel processing of said packet stream.

14. The system of claim 11 wherein said compiler is further configured to generate said module in part by processing said received replacement string to thereby create a logical representation of a string replacement machine therefrom that is operable to replace a matching string found in data with said received replacement string, said module including said string replacement machine representation.

15. The system of claim 14 wherein said compiler is further configured generate said module in part by coordinating said pattern matching state machine representation and said string replacement machine representation with a logical representation of a controller, said controller representation being operable to (1) communicate with said pattern matching state machine representation to determine a start position and an end position of a matching string, and (2) process said determined start and end positions to control said string replacement machine representation, said module including said controller representation.

16. The system of claim 4 wherein said PLD is a field programmable gate array (FPGA).

17. The system of claim 4 wherein said data pattern encompasses at least a word in a first language, and wherein said replacement string comprises a translation of said at least one word in said first language into a second language.

18. The system of claim 4 wherein said data pattern encompasses at least in part a profanity, and wherein said replacement string comprises a data string not including said profanity.

19. The system of claim 4 wherein said data pattern encompasses an encrypted data string, and wherein said replacement string comprises a data string corresponding to a decryption of said encrypted data string.

20. The system of claim 4 wherein said data pattern encompasses a data string, and wherein said replacement string comprises an encryption of said data string.

21. The system of claim 4 wherein said data pattern encompasses at least a portion of a computer virus, and wherein said replacement string comprises a data string that is not a computer virus.

22. The system of claim 3 wherein said string replacement operation is a back substitution operation.

23. The system of claim 3 wherein said data processor is in communication with a computer network from which said data stream is received, said data stream comprising a stream of data packets transmitted over said computer network, wherein said redefinable action is a packet drop operation, and wherein said PLD is programmed to perform said packet drop operation by dropping a packet that is found to include a matching string.

24. The system of claim 1 wherein said redefinable action is a notification operation, wherein said PLD is programmed to perform said notification operation by sending a notification signal to an interested device, said notification signal being operative to identify the existence of a matching string in said data stream.

25. The system of claim 24 wherein said data processor is in communication with a computer network from which said data stream is received, said data stream comprising a stream of data packets transmitted over said computer network, wherein said notification signal is a notification packet addressed for transmission to an

interested party, wherein said notification packet includes a copy of the packet that includes said matching string.

26. The system of claim 1 wherein said redefinable action is an awk operation, and wherein said PLD is programmed to perform said awk operation when a matching string is found in said data stream.

27. The system of claim 1 wherein said data pattern encompasses at least a portion of at least one of the group consisting of an image file, an audio file, a video file, an audio/video file, software, virus infected file, text file, and electronic publishing files.

28. The system of claim 1 wherein said data pattern encompasses at least a portion of a copyright-protected work.

29. The system of claim 1 wherein said data pattern encompasses at least a portion of a trade secret.

30. The system of claim 1 wherein said data pattern encompasses a data string indicative of a criminal conspiracy.

31. A method of processing a stream of data, said method comprising:

programming a programmable logic device (PLD) to (1) determine whether a stream of data applied thereto includes a string that matches a data pattern, and (2) perform a responsive action if said data stream is found to include a matching string;

processing a stream of data through said programmed PLD to (1) determine whether said data stream includes a string that matches said data pattern, and (2) perform said responsive action if said data stream is found to include a matching string; and

repeating said programming step with at least one of the group consisting of a different data pattern and a different action.

32. The method of claim 31 wherein said programming step includes:

receiving a data pattern;

receiving an action command, said action command specifying an action to be performed if said data stream is found to include a matching string;

generating configuration information from said received data pattern and said received action command that defines a module that is operable upon being programmed into said PLD to (1) determine whether a stream of data applied thereto includes a string that matches said received data pattern, and (2) perform said responsive action if said data stream is found to include a matching string; and

programming said PLD with said configuration information.

33. The method of claim 32 wherein said action command receiving step includes receiving an action command that specifies a data modification operation, said data modification operation identifying how a data stream is to be modified if that data stream is found to include a matching string.

34. The method of claim 33 wherein said action command receiving step includes receiving an action command that specifies a string replacement operation, said string replacement operation identifying a replacement string to be inserted into a data stream in place of a matching string.

35. The method of claim 32 wherein said action command receiving step includes receiving an action command wherein said action command specifies a notification operation, said notification operation specifying a notification signal to be transmitted when a matching string is found in said data stream.

36. The method of claim 32 wherein said data stream is a stream of packets transmitted over a computer network, each data packet including a payload portion, and wherein said processing step includes:

receiving a stream of data packets;

identifying the payload portion of each received data packet; and

processing the payload portion of each received data packet through said programmed PLD to (1) determine whether the payload portion of any received data packet includes a string that matches said data pattern, and (2) perform said responsive action if said a payload portion is found to include a matching string.

37. The method of claim 36 wherein said step of programming said PLD with said configuration information includes transmitting said configuration information over said computer network to said PLD.

38. A device for generating configuration information operable to program a programmable logic device (PLD) with a data processing module operable to receive and process a stream of data to determine whether said data stream includes a data pattern and, if so, perform a responsive action, said device comprising:

an input operable to receive a data pattern and an action command from a user, said action command specifying an action to be performed if said data stream is found to include a string that matches said data pattern;

a compiler operable to generate configuration information from said received data pattern and said received modification command, said configuration information defining a data processing module operable upon being programmed into said PLD to (1) determine whether a stream of data applied thereto includes a string that matches said received data pattern, and (2) perform said action if said data stream is found to include a matching string, said configuration information being operable to program said PLD with said data processing module.

39. The device of claim 38 wherein said action command is a modification command, said modification command specifying a modification operation to be performed upon said data stream if said data stream is found to include a matching string, and wherein said compiler is also operable to generate said configuration information from said modification command such that said module defined thereby is also operable upon being programmed into said PLD to perform said modification operation upon said data stream if said data stream is found to include a matching string.

40. The device of claim 39 wherein said compiler is also operable to process said received data pattern through a lexical analyzer generator to thereby generate a logical representation of a pattern matching state machine that is operable to determine whether a stream of data applied thereto includes a string that matches said received

data pattern, said pattern matching state machine representation in part defining said module.

41. The device of claim 40 wherein said modification operation specified by said modification command is a string replacement operation, wherein said modification command includes a replacement string, and wherein said compiler is also operable to process said received modification command to thereby generate a logical representation of a string replacement machine that is operable to replace a matching string in said data stream with said replacement string, said string replacement machine representation in part defining said module.

42. The device of claim 41 wherein said compiler is also operable to process said received data pattern through said lexical analyzer generator such that said pattern matching state machine representation comprises at least one deterministic finite automaton (DFA).

43. The device of claim 42 wherein said data stream is a packet stream comprising a plurality of data packets transmitted over a computer network, said device further comprising a transmitter interfacing said compiler with said computer network, said transmitter being operable to receive said configuration information from said compiler and transmit said configuration information over said network to a programming device in communication with said PLD, said programming device being operable to program said PLD with said module defined thereby.

44. The device of claim 43 wherein each packet in said packet stream includes a payload portion, and wherein said compiler is further operable to generate said configuration information such that said module defined thereby is also operable upon being programmed into said PLD to determine whether any of said payloads of said packets comprising said packet stream include a matching string.

45. The device of claim 44 wherein said input is also operable to receive said data pattern in a regular expression format.

46. The device of claim 45 wherein said input is also operable to receive a plurality of data patterns and a plurality of said modification commands from a user, each modification command having a corresponding data pattern, and wherein said compiler is also operable to generate said configuration information such that said data processing module defined thereby is also operable to, for each data pattern and corresponding modification command, perform said match determination and said string replacement operation.

47. A method of programming a programmable logic device (PLD) to process a stream of data, said method comprising:

receiving a data pattern;

receiving a modification command corresponding to said data pattern, said action command specifying an action to be performed if said stream of data is found to include a string that matches said data pattern;

generating configuration information from said received data pattern and said received modification command that is operable to program said PLD with a data processing module that is operable upon being programmed into said PLD to (1) determine whether a data stream applied thereto includes a string that matches said data pattern, and (2) perform said action specified by said action command if said data stream is found to include a matching string; and

communicating said configuration information to a programming device in communication with said PLD, said programming device being operable to program said PLD with said configuration information.

48. The method of claim 47 wherein said action command is a modification command that specifies a modification operation to performed upon a data stream that includes a matching string, and wherein said generating step includes generating said configuration information such that said module defined thereby is also operable to perform said modification operation upon said data stream if said data stream is found to include a matching string.

49. The method of claim 48 wherein said modification operation is a string replacement operation, wherein said modification command includes a replacement string, and wherein said generating step includes generating said configuration information such that said

module defined thereby is also operable to perform said string replacement operation by replacing a matching string in said data stream with said replacement string.

50. The method of claim 49 wherein said generating step includes processing said data pattern through a lexical analyzer generator to create a logical representation of a pattern matching state machine therefrom, said pattern matching state machine in part defining said module and being operable to determine whether a stream of data applied thereto includes a string that matches said data pattern.

51. The method of claim 50 wherein said generating step includes creating a logical representation of a string replacement machine from said received modification command, said string replacement machine in part defining said module and being operable to replace a matching string in said data stream with said replacement string.

52. The method of claim 51 wherein said data stream is a packet stream comprising a plurality of data packets transmitted over a computer network, and wherein said communicating step includes communicating said configuration information over said network to said programming device.

53. The method of claim 52 wherein each of said data packets includes a payload portion, and wherein said generating step includes generating said configuration information such that said module defined thereby is also operable upon being programmed into said PLD to determine whether any of said payload portions of said packets comprising said packet stream include a matching string.

54. The method of claim 53 wherein said data pattern receiving step includes receiving said data pattern in a regular expression format.

55. The method of claim 54 wherein said data pattern receiving step includes receiving a plurality of data patterns, wherein said modification command receiving step includes receiving a plurality of said modification commands, each modification command corresponding to a data pattern, and wherein said generating step includes generating said configuration information such that said module

defined thereby is also operable to, for each data pattern and corresponding modification command, (1) perform said match determination, and (2) perform said string replacement operation

56. A device for processing a stream of data, said device comprising:

a programmable logic device (PLD) programmed to receive a stream of data and process said data stream through a plurality of redefinable logic structures in series, each logic structure being tuned with a corresponding redefinable data pattern and being operable to determine whether a string is present in said processed data stream that matches that logic structure's corresponding data pattern.

57. The device of claim 56 wherein each logic structure is also tuned with a corresponding redefinable action and is further operable to perform that logic structure's corresponding redefinable action if said processed data stream is found to include a string that matches that logic structure's corresponding data pattern.

58. The device of claim 57 wherein each redefinable action is a string replacement operation, each string replacement operation including a replacement string, and wherein each logic structure is further operable to replace a string found in said processed data stream that matches that logic structure's corresponding data pattern with that logic structure's corresponding replacement string.

59. The device of claim 58 wherein said PLD is a field programmable gate array (FPGA) in communication with a computer network, said data stream comprising a stream of data packets transmitted over said computer network.

60. A device of processing a stream of data, said device comprising:

a programmable logic device (PLD) programmed to receive a stream of data and process said received data stream through a plurality of pattern matching state machines in parallel, each pattern matching state machine of said plurality of pattern matching state machines being tuned with a data pattern and being operable to

determine whether said data stream includes a string that matches the data pattern with which it is tuned.

61. The device of claim 60 wherein each pattern matching state machine of said plurality of pattern matching state machines is tuned with the same data pattern.

62. The device of claim 61 wherein said data stream comprises a stream of data bytes, wherein said PLD is also programmed with a controller operable to provide said data stream to said plurality of parallel pattern matching state machines such that each pattern matching state machine receives said data stream starting at a different byte.

63. The device of claim 62 wherein said controller is also operable to communicate with said plurality of parallel pattern matching state machines to identify a longest string in said data stream that matches said data pattern.

64. The device of claim 60 each pattern matching state machine of said plurality of pattern matching state machines is tuned with a different data pattern.

65. A reprogrammable system for processing a stream of data, said system comprising:

a reprogrammable data processor for receiving a stream of data and processing said received data stream through a programmable logic device (PLD) programmed with at least one deterministic finite automaton (DFA) to determine whether said data stream includes a string that matches a redefinable data pattern; and

a reconfiguration device in communication with said data processor that is operable to reprogram said PLD with a different DFA to determine whether a data stream includes a string that matches a different data pattern.

66. A network processor for processing a stream of data packets transmitted over a computer network, said network processor comprising:

a protocol wrapper operative to receive data from said computer network and process said data to generate a stream of data packets therefrom, said packet stream comprising a stream of words, each word including a plurality of bytes;

a matching path operative to receive said packet stream from said protocol wrapper and detect whether any of said packets comprising said packet stream include a string that matches a data pattern;

a controller in communication with said matching path that is operative to determine a starting byte position and an ending byte position of each matching string detected by said matching path;

a data path in communication with said controller that is operative to receive said packet stream from said protocol wrapper and process each starting byte position and ending byte position for each matching string determined by said controller to (1) output each byte of said packet stream that does not correspond to a matching string, and (2) replace the bytes of said packet stream that correspond to a matching string with a replacement string;

wherein said matching path, said controller, and said data path are implemented on a programmable logic device (PLD).

67. The network processor of claim 66 wherein said protocol wrapper is also implemented on said PLD.

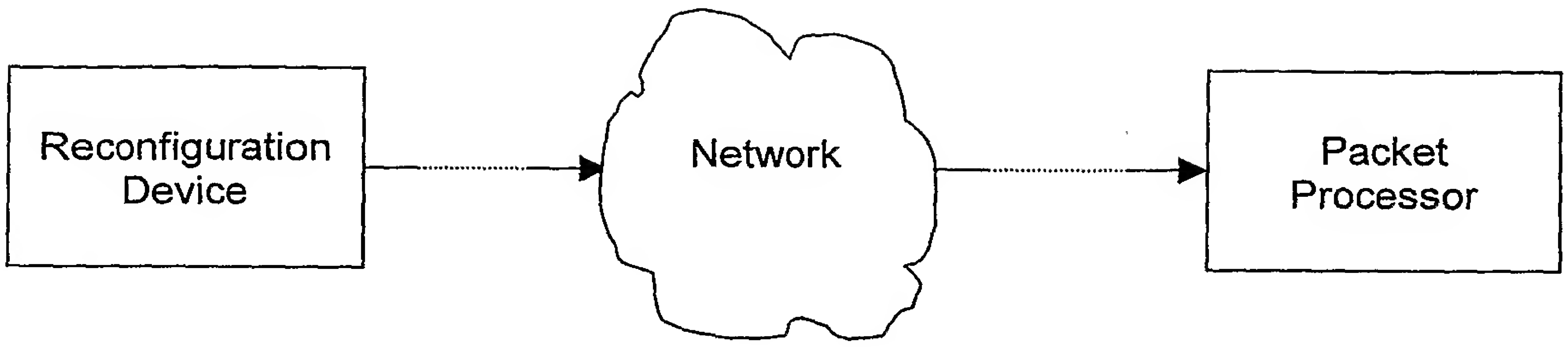


Figure 1(a)

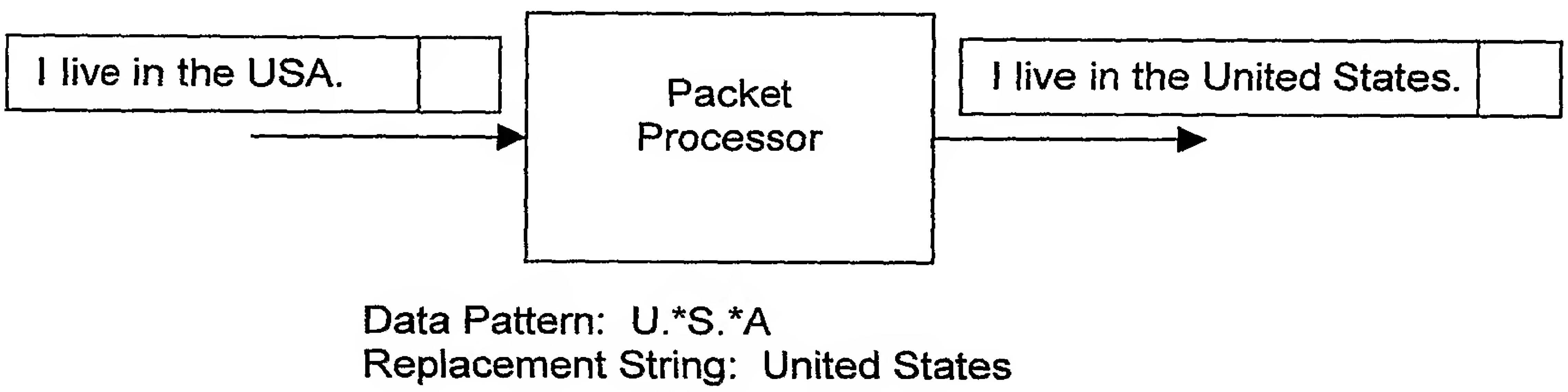


Figure 1(b)

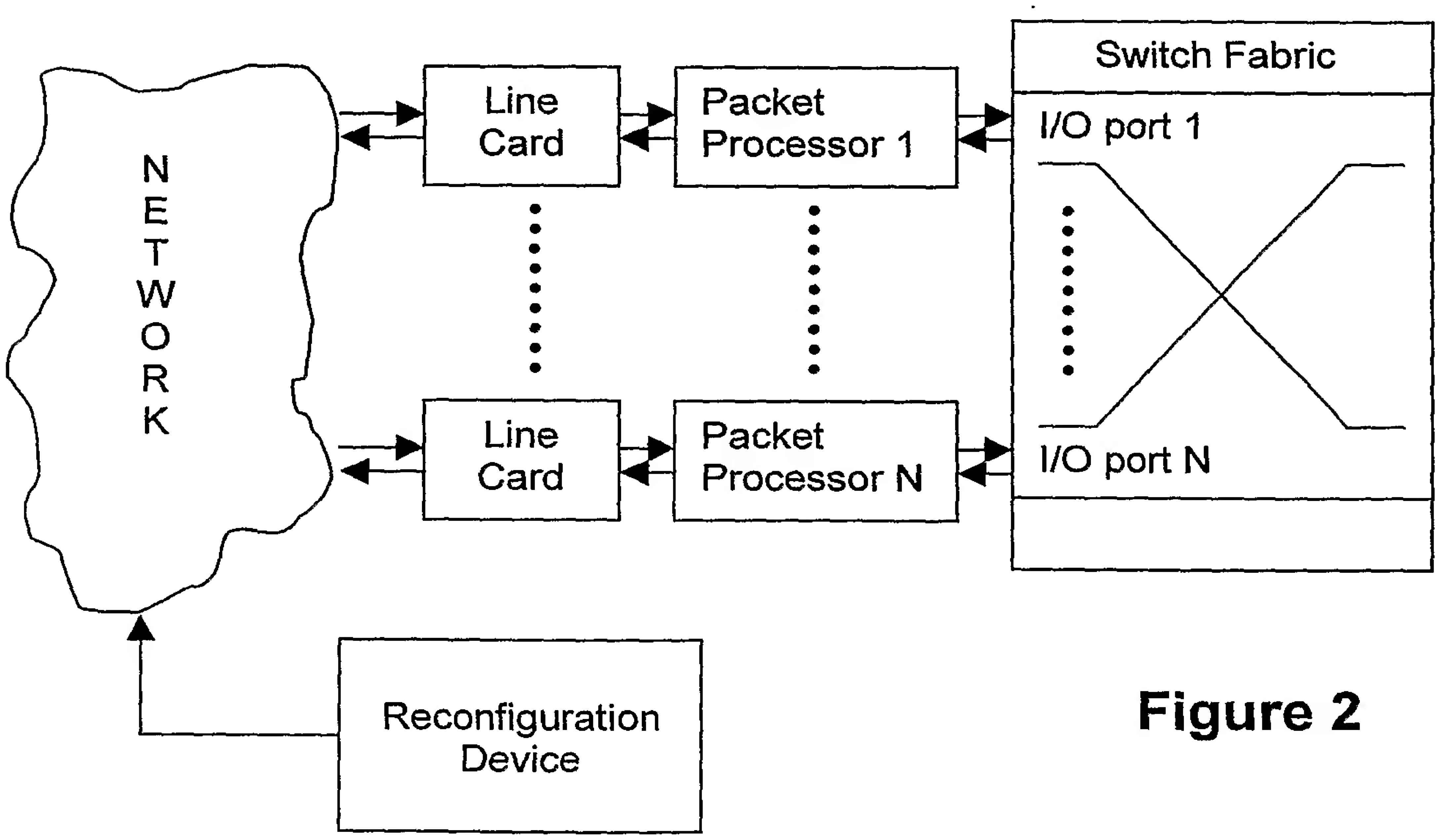


Figure 2

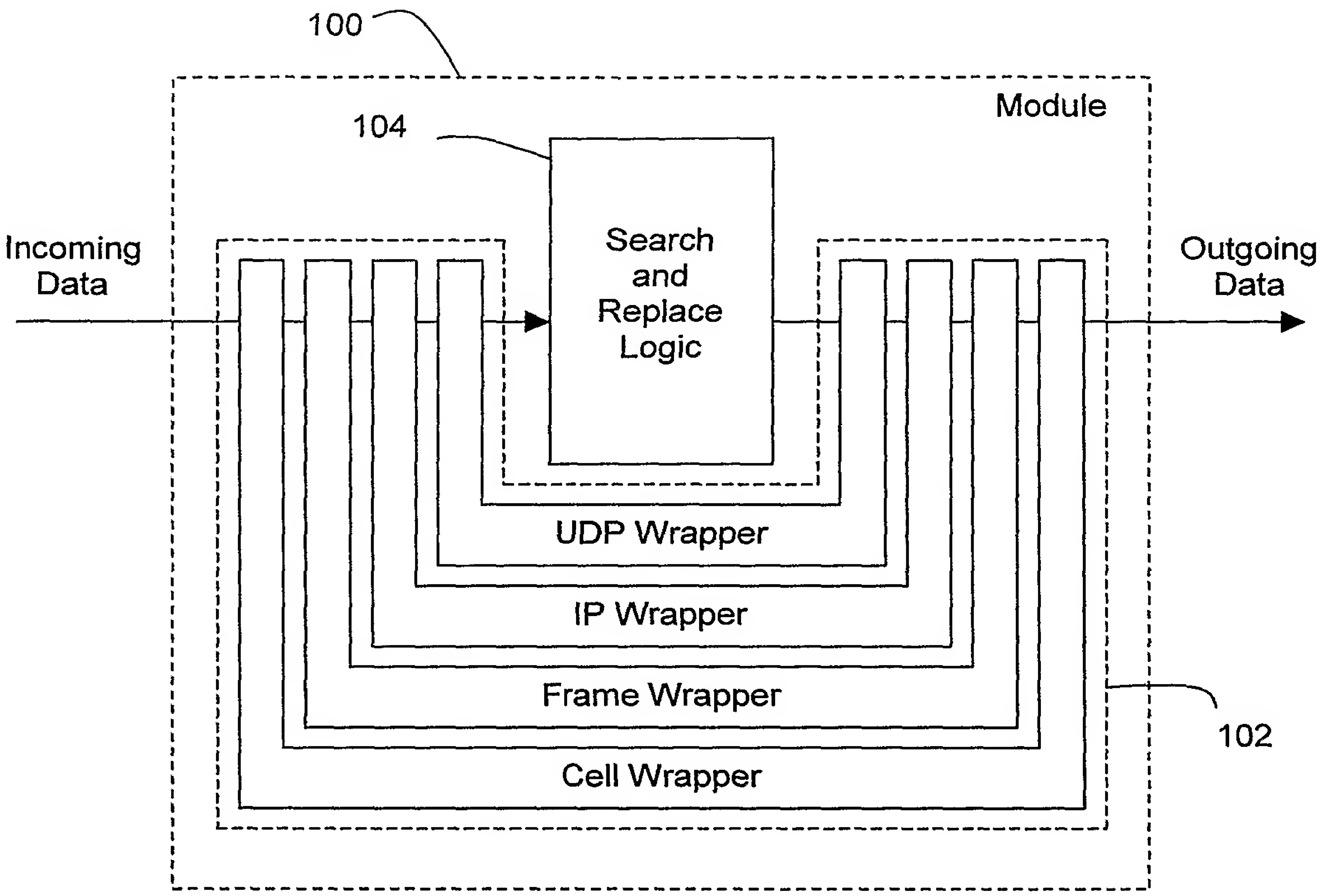


Figure 3

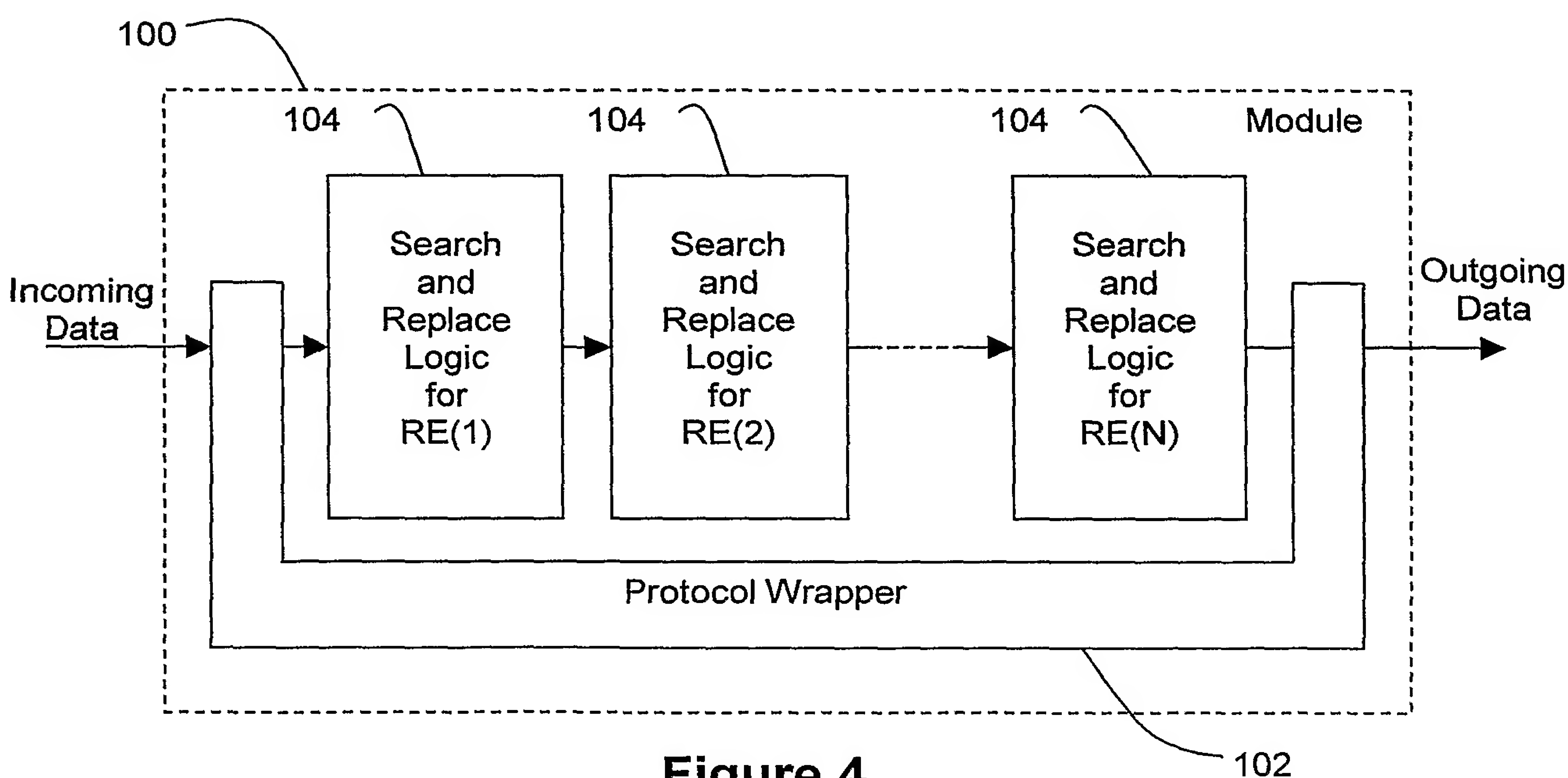
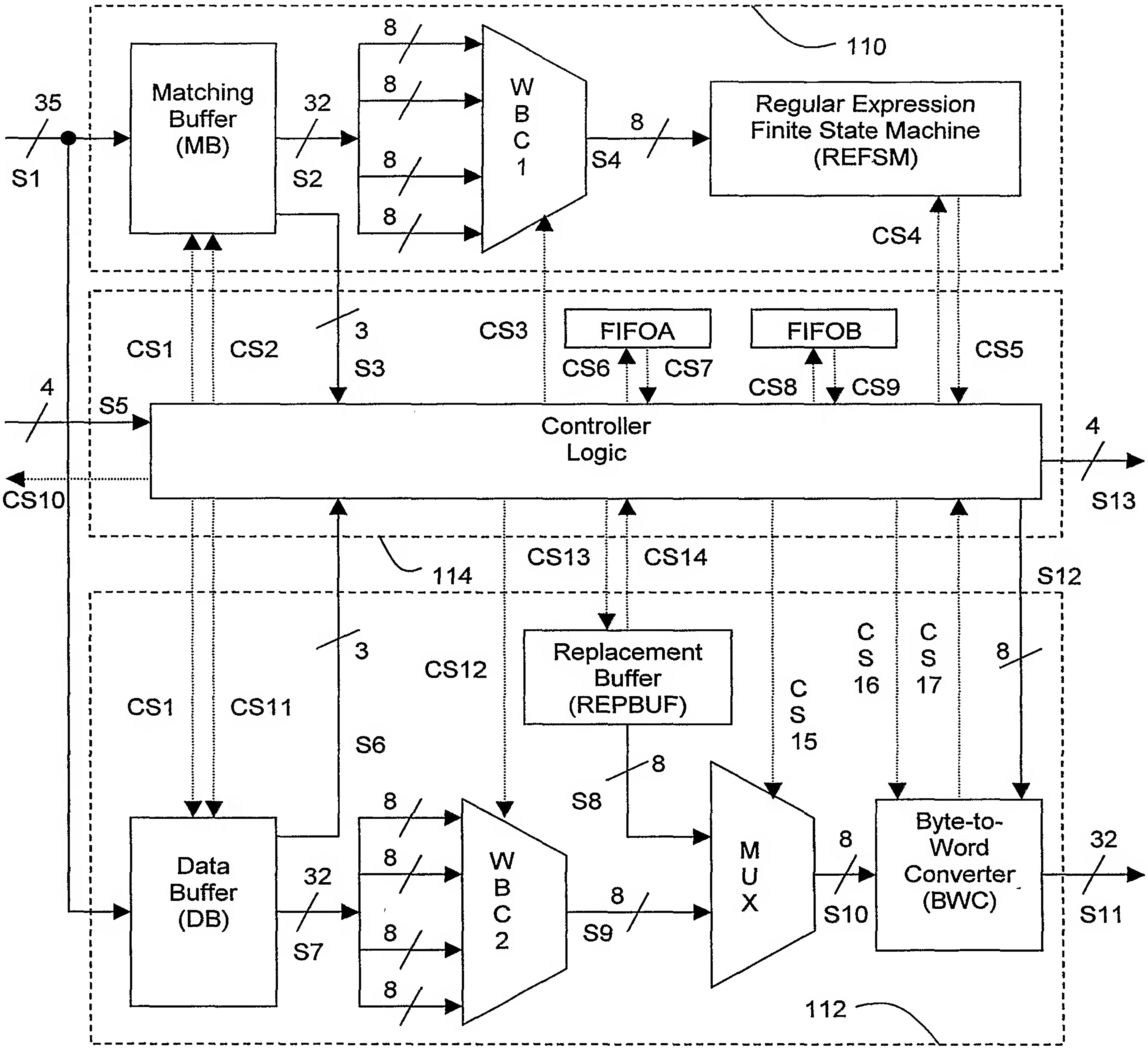


Figure 4



Signal Table

S1:	32 Bit Word Input+SOF, SOD, EOF	**CS1:**	DATA_WR_ADD	**CS10:**	CONGESTION
S2:	32 Bit Word from MB to WBC1	**CS2:**	MB_RD_ADD	**CS11:**	DB_RD_ADD
S3:	Matching Path Control Bits	**CS3:**	WBC1_SELECT	**CS12:**	WBC2_SELECT
S4:	Byte from WBC1 to REFSM	**CS4:**	REFSM_ENABLE	**CS13:**	
	REPBUF_ENABLE				
S5:	4 Bit Control Information Input	**CS5:**	REFSM_STATUS	**CS14:**	REPBUF_DONE
S6:	Data Path Control Bits	**CS6:**	WR_START_ADD	**CS15:**	MUX_SELECTOR
S7:	32 Bit Word from DB to WBC2	**CS7:**	RD_START_ADD	**CS16:**	BWC_ENABLE
S8:	Byte of Replacement String	**CS8:**	WR_END_ADD	**CS17:**	
	PADDING_COUNT				
S9:	Byte from WBC2	**CS9:**	RD_END_ADD		
S10:	Byte from MUX output				
S11:	32 Bit Word Output				
S12:	Padding Byte				
S13:	4 Bit Control Information Output				

Figure 5

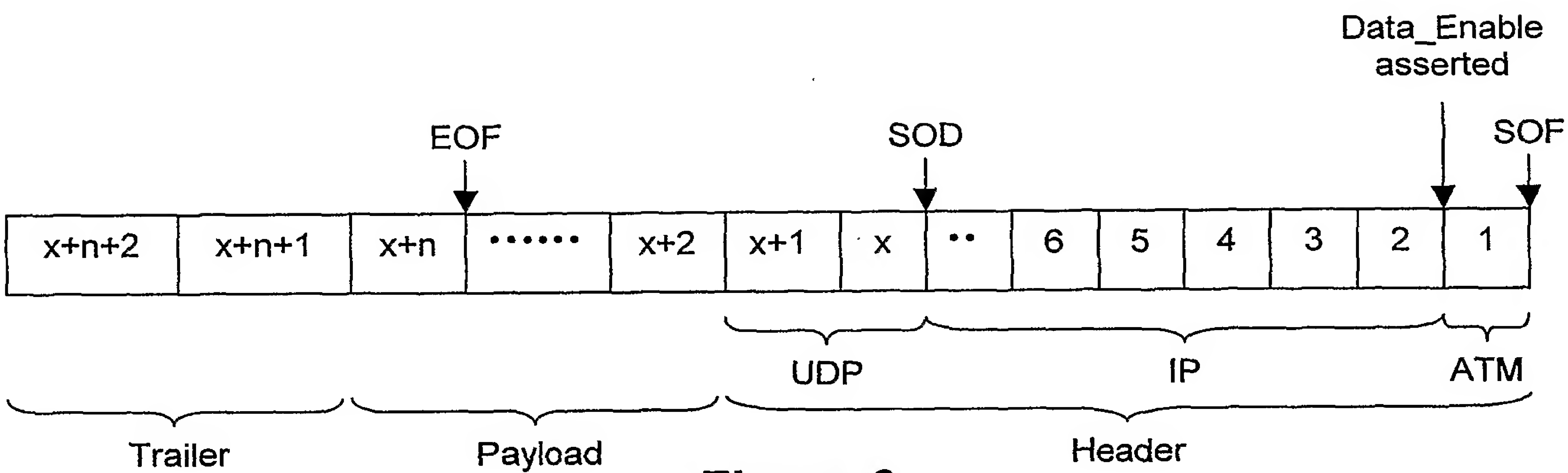


Figure 6

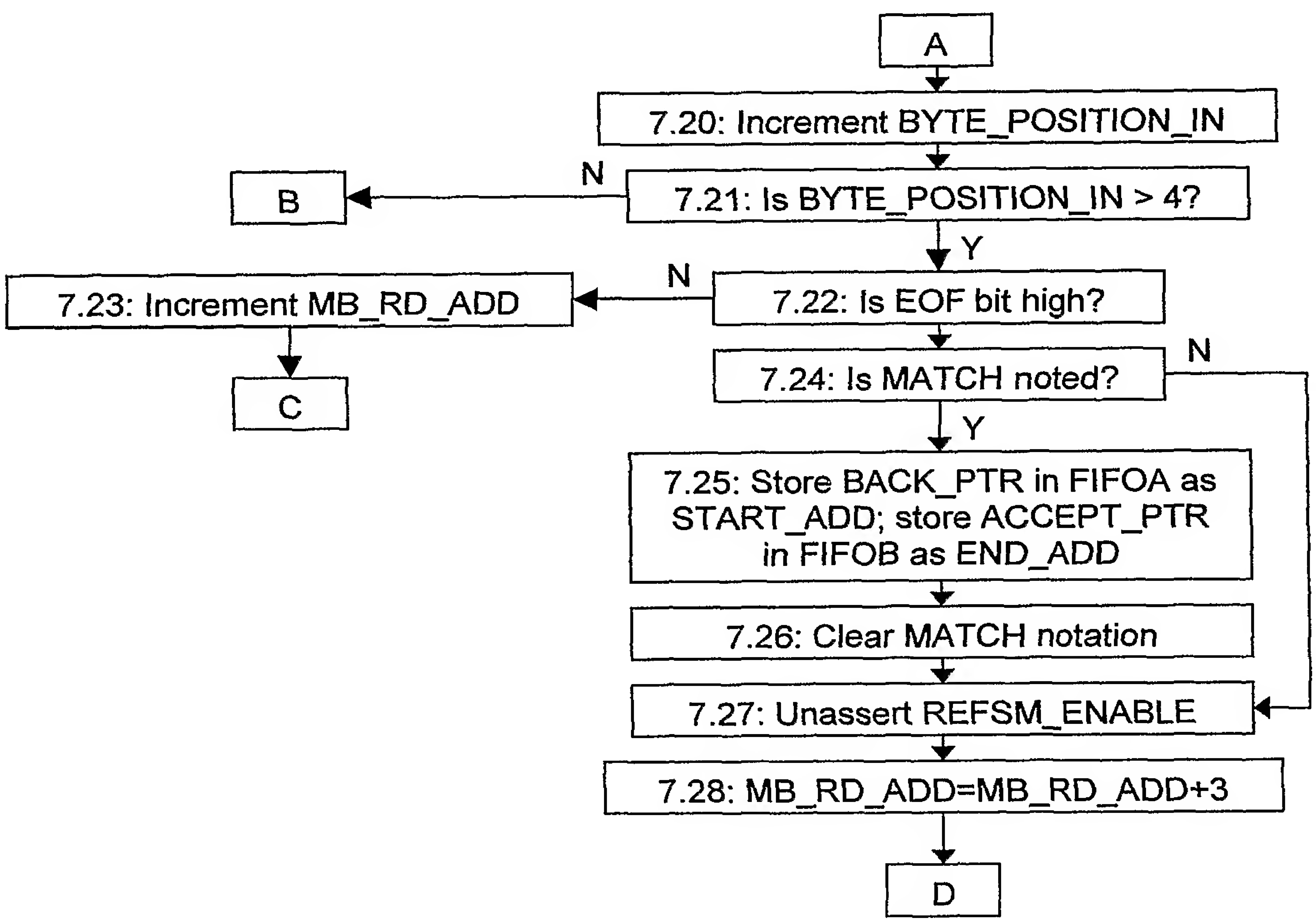


Figure 7(b)

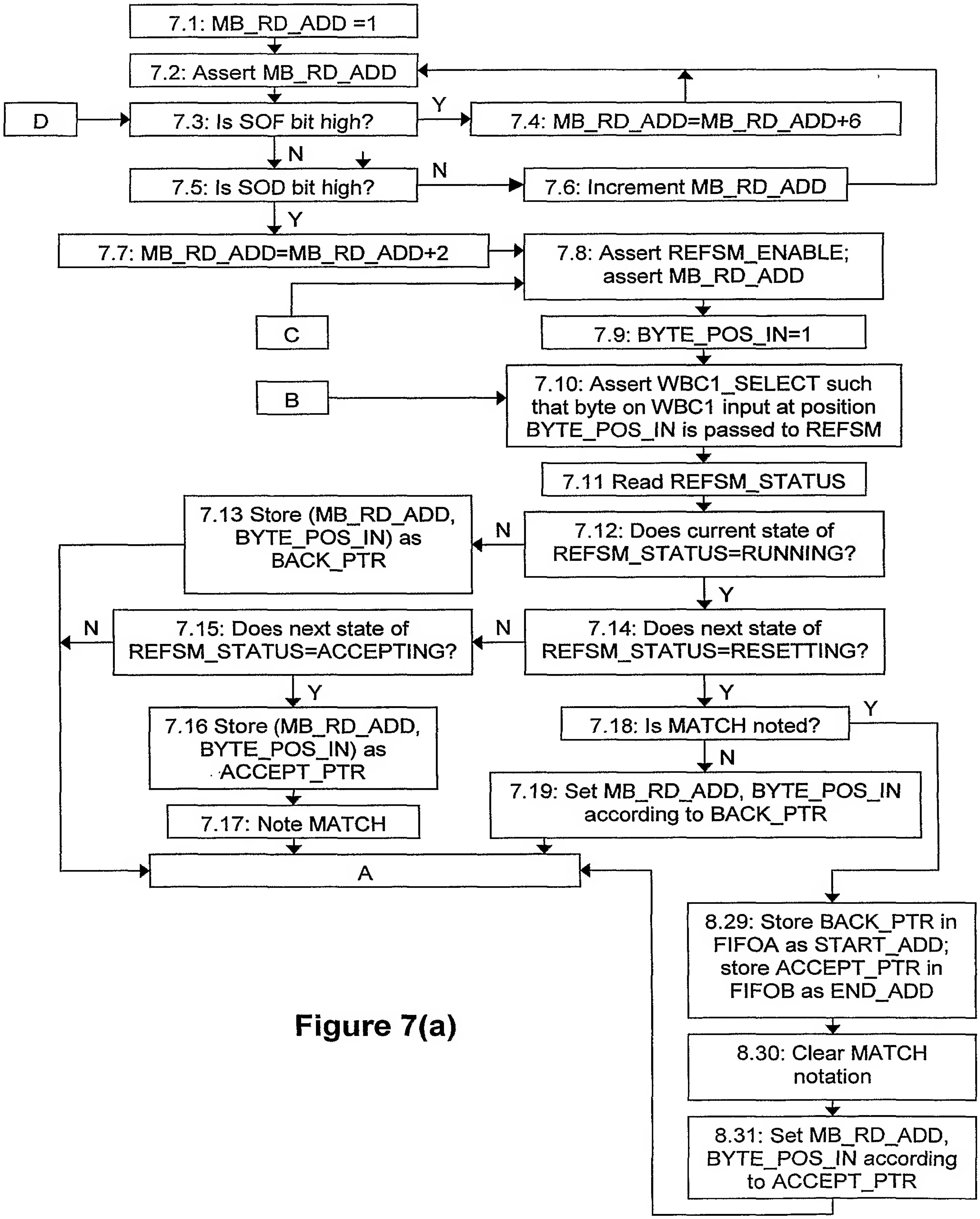

7.1: MB_RD_ADD =1
7.2: Assert MB_RD_ADD
D
7.3: Is SOF bit high?
Y
7.4: MB_RD_ADD=MB_RD_ADD+6
N
7.5: Is SOD bit high?
N
7.6: Increment MB_RD_ADD
Y
7.7: MB_RD_ADD=MB_RD_ADD+2
7.8: Assert REFSM_ENABLE; assert MB_RD_ADD
C
7.9: BYTE_POS_IN=1
B
7.10: Assert WBC1_SELECT such that byte on WBC1 input at position BYTE_POS_IN is passed to REFSM
7.11 Read REFSM_STATUS
7.13 Store (MB_RD_ADD, BYTE_POS_IN) as BACK_PTR
N
7.12: Does current state of REFSM_STATUS=RUNNING?
Y
N
7.15: Does next state of REFSM_STATUS=ACCEPTING?
N
7.14: Does next state of REFSM_STATUS=RESETTING?
Y
Y
7.16 Store (MB_RD_ADD, BYTE_POS_IN) as ACCEPT_PTR
7.18: Is MATCH noted?
Y
N
7.19: Set MB_RD_ADD, BYTE_POS_IN according to BACK_PTR
7.17: Note MATCH
A
8.29: Store BACK_PTR in FIFOA as START_ADD; store ACCEPT_PTR in FIFOB as END_ADD
8.30: Clear MATCH notation
8.31: Set MB_RD_ADD, BYTE_POS_IN according to ACCEPT_PTR

Figure 7(a)

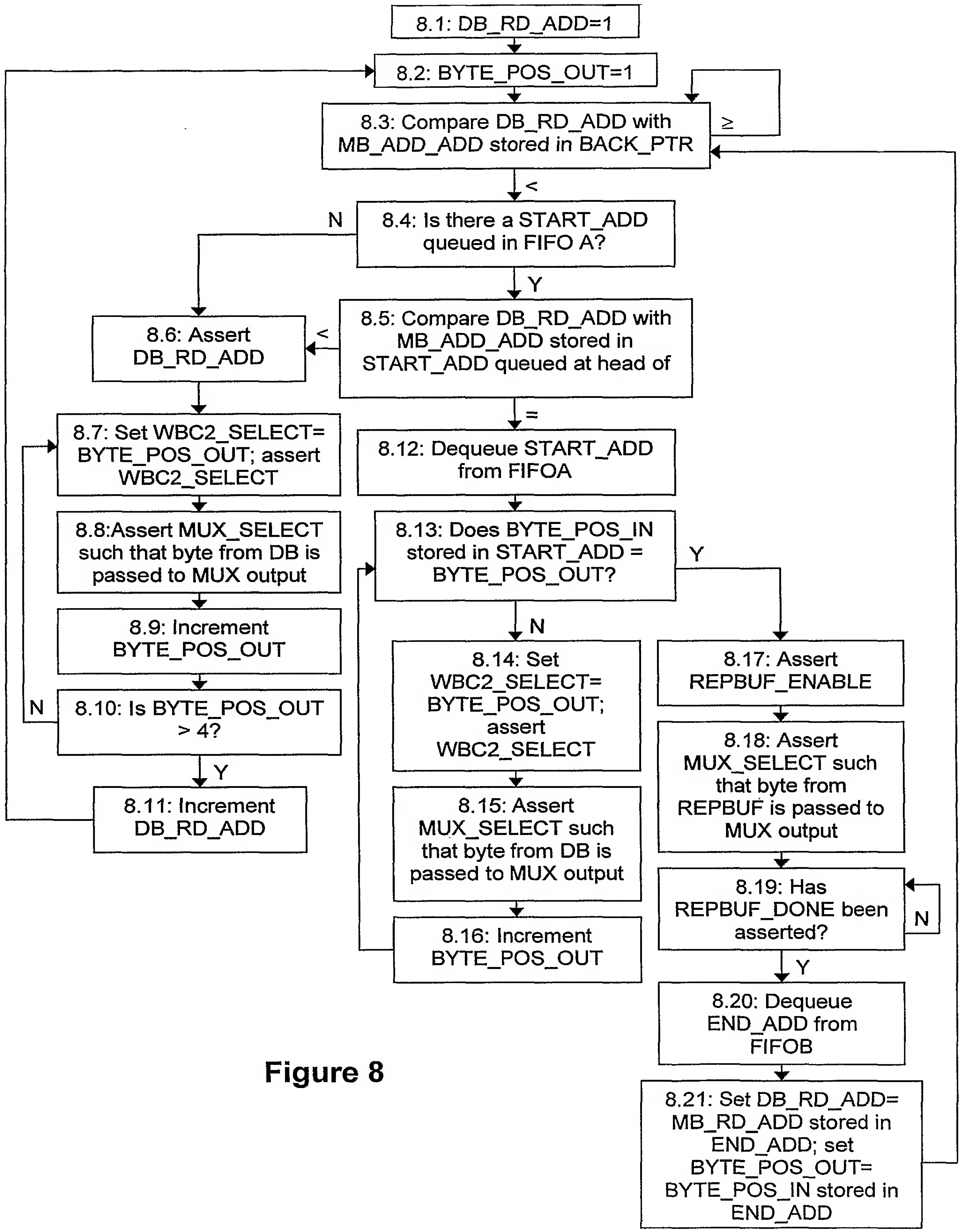

8.1: DB_RD_ADD=1
8.2: BYTE_POS_OUT=1
8.3: Compare DB_RD_ADD with MB_ADD_ADD stored in BACK_PTR
≥
<
8.4: Is there a START_ADD queued in FIFO A?
N
Y
8.5: Compare DB_RD_ADD with MB_ADD_ADD stored in START_ADD queued at head of
8.6: Assert DB_RD_ADD
<
=
8.7: Set WBC2_SELECT= BYTE_POS_OUT; assert WBC2_SELECT
8.12: Dequeue START_ADD from FIFOA
8.8:Assert MUX_SELECT such that byte from DB is passed to MUX output
8.13: Does BYTE_POS_IN stored in START_ADD = BYTE_POS_OUT?
Y
8.9: Increment BYTE_POS_OUT
N
8.14: Set WBC2_SELECT= BYTE_POS_OUT; assert WBC2_SELECT
8.17: Assert REPBUF_ENABLE
N
8.10: Is BYTE_POS_OUT > 4?
Y
8.18: Assert MUX_SELECT such that byte from REPBUF is passed to MUX output
8.11: Increment DB_RD_ADD
8.15: Assert MUX_SELECT such that byte from DB is passed to MUX output
8.19: Has REPBUF_DONE been asserted?
N
8.16: Increment BYTE_POS_OUT
Y
8.20: Dequeue END_ADD from FIFOB
8.21: Set DB_RD_ADD= MB_RD_ADD stored in END_ADD; set BYTE_POS_OUT= BYTE_POS_IN stored in END_ADD

Figure 8

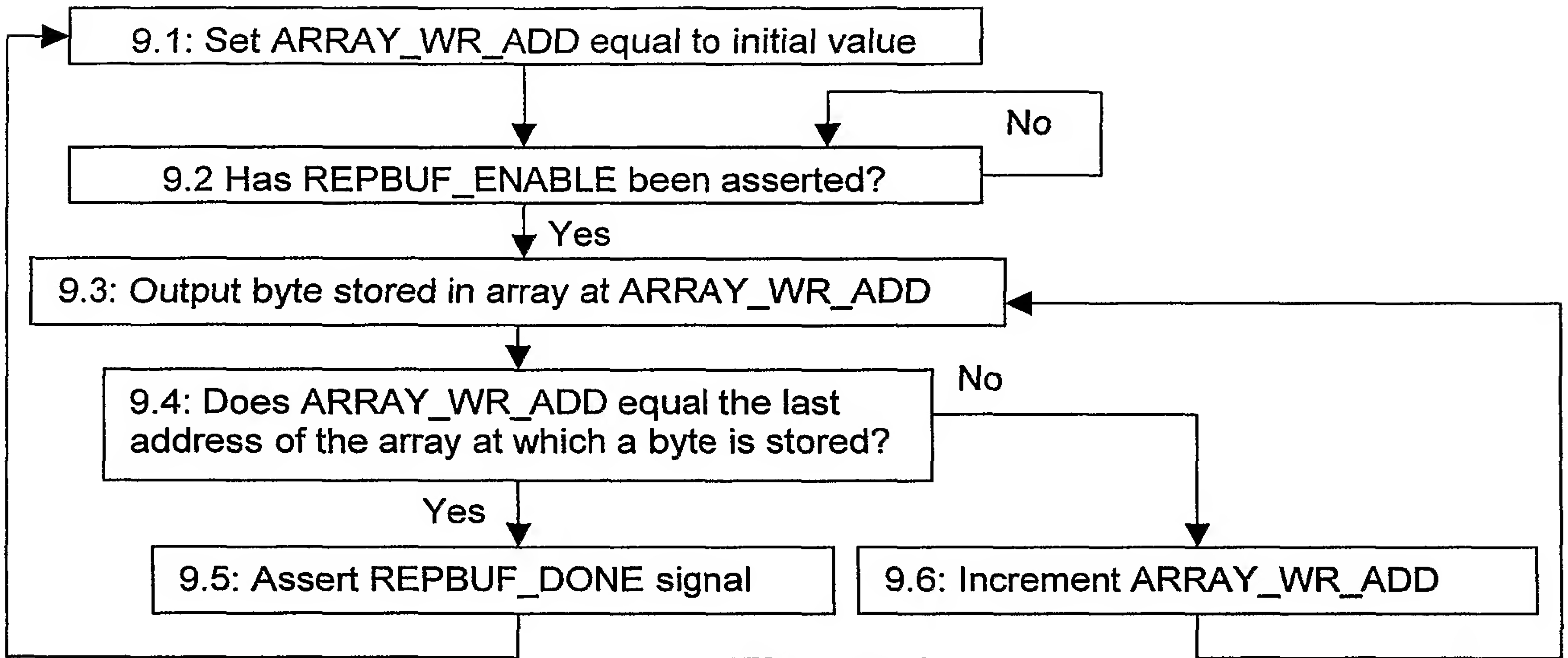


Figure 9

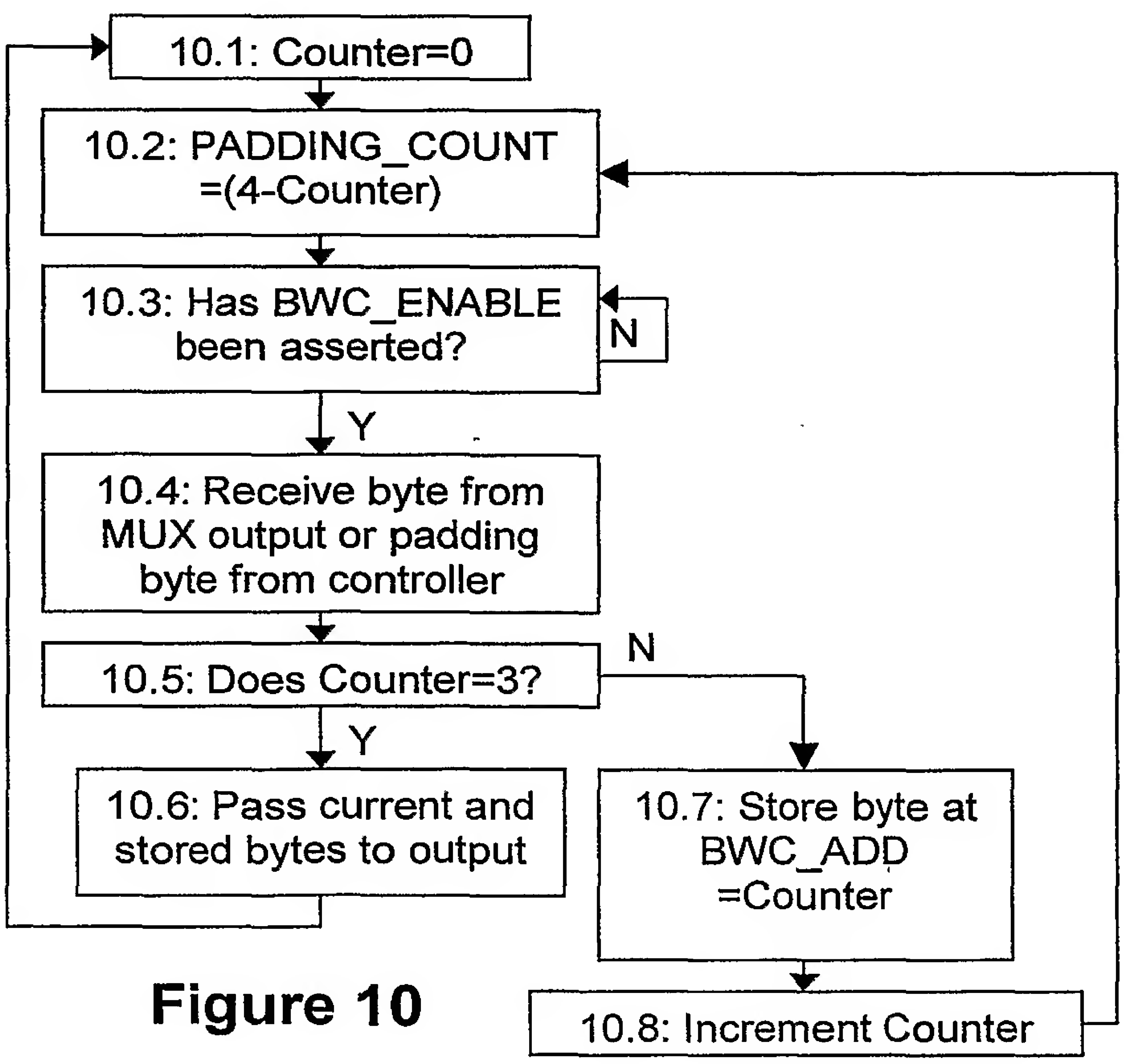


Figure 10

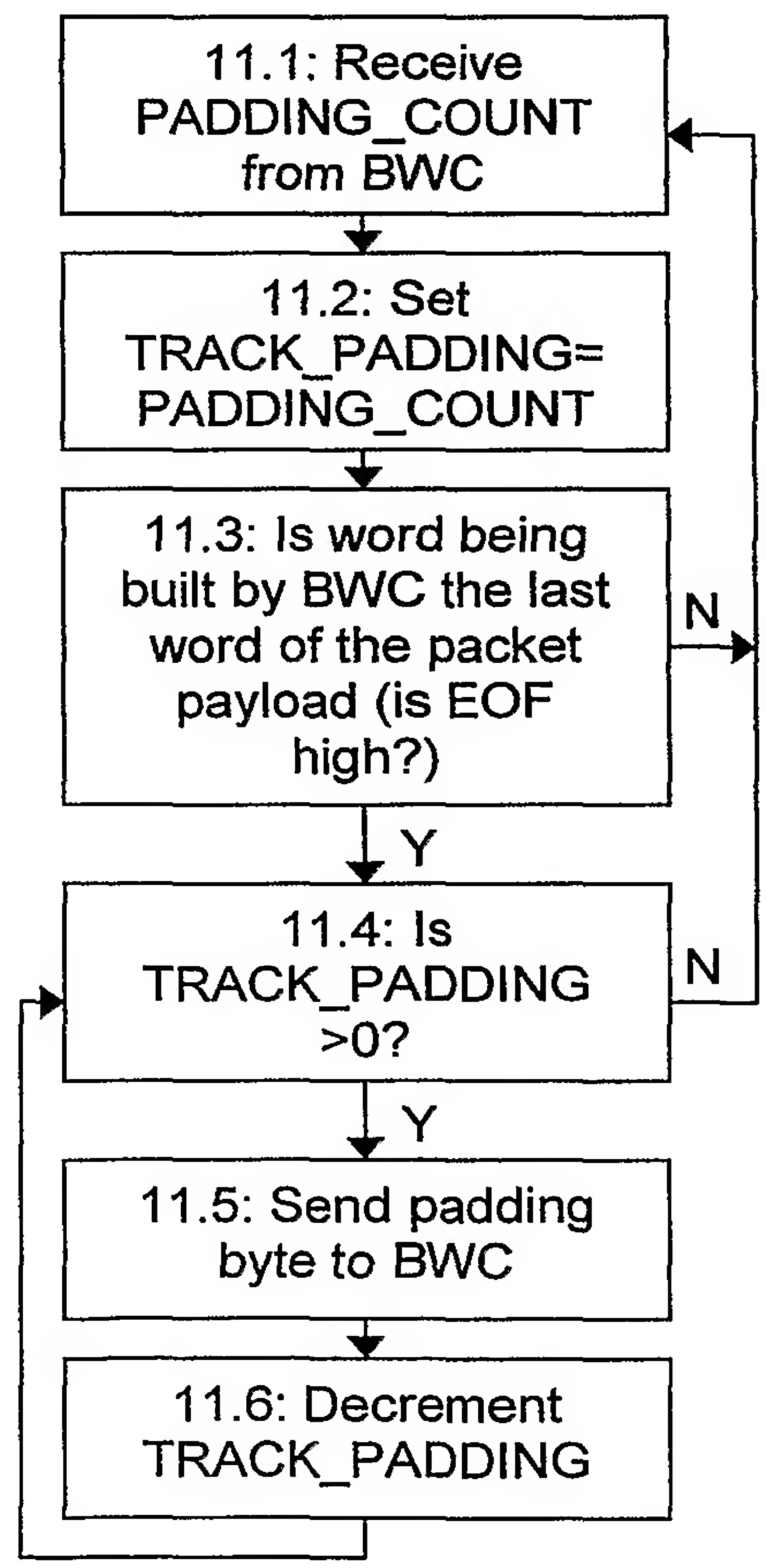


Figure 11

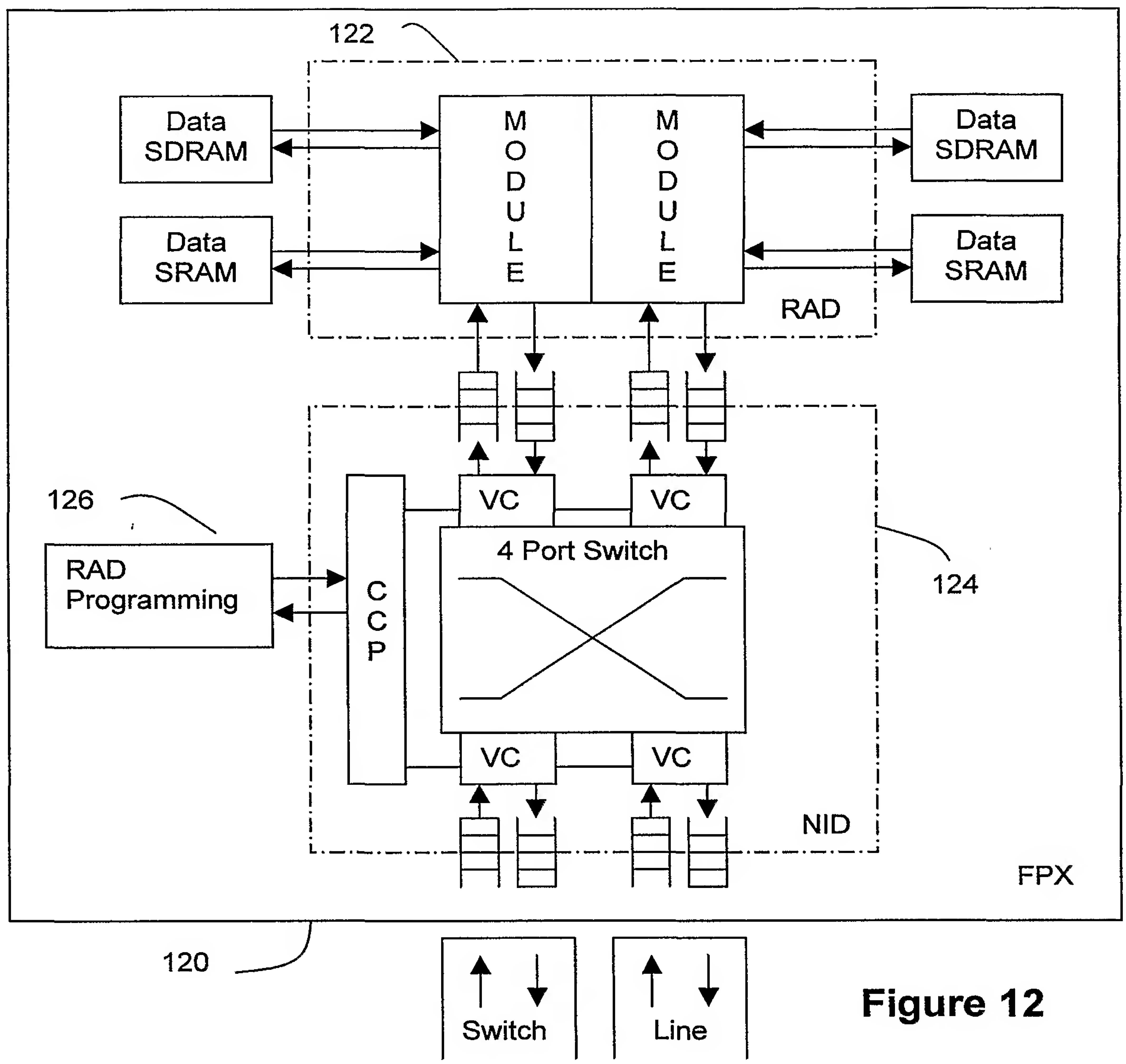

122
Data SDRAM
Data SRAM
MODULE
MODULE
Data SDRAM
Data SRAM
RAD
126
RAD Programming
CCP
VC
VC
4 Port Switch
VC
VC
NID
124
FPX
120
Switch
Line

Figure 12

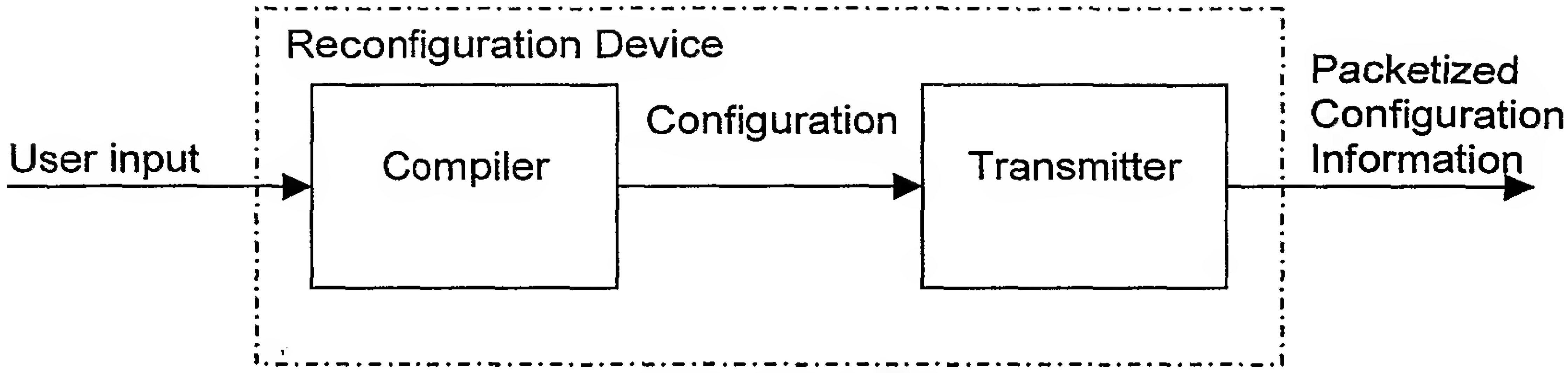


Figure 13

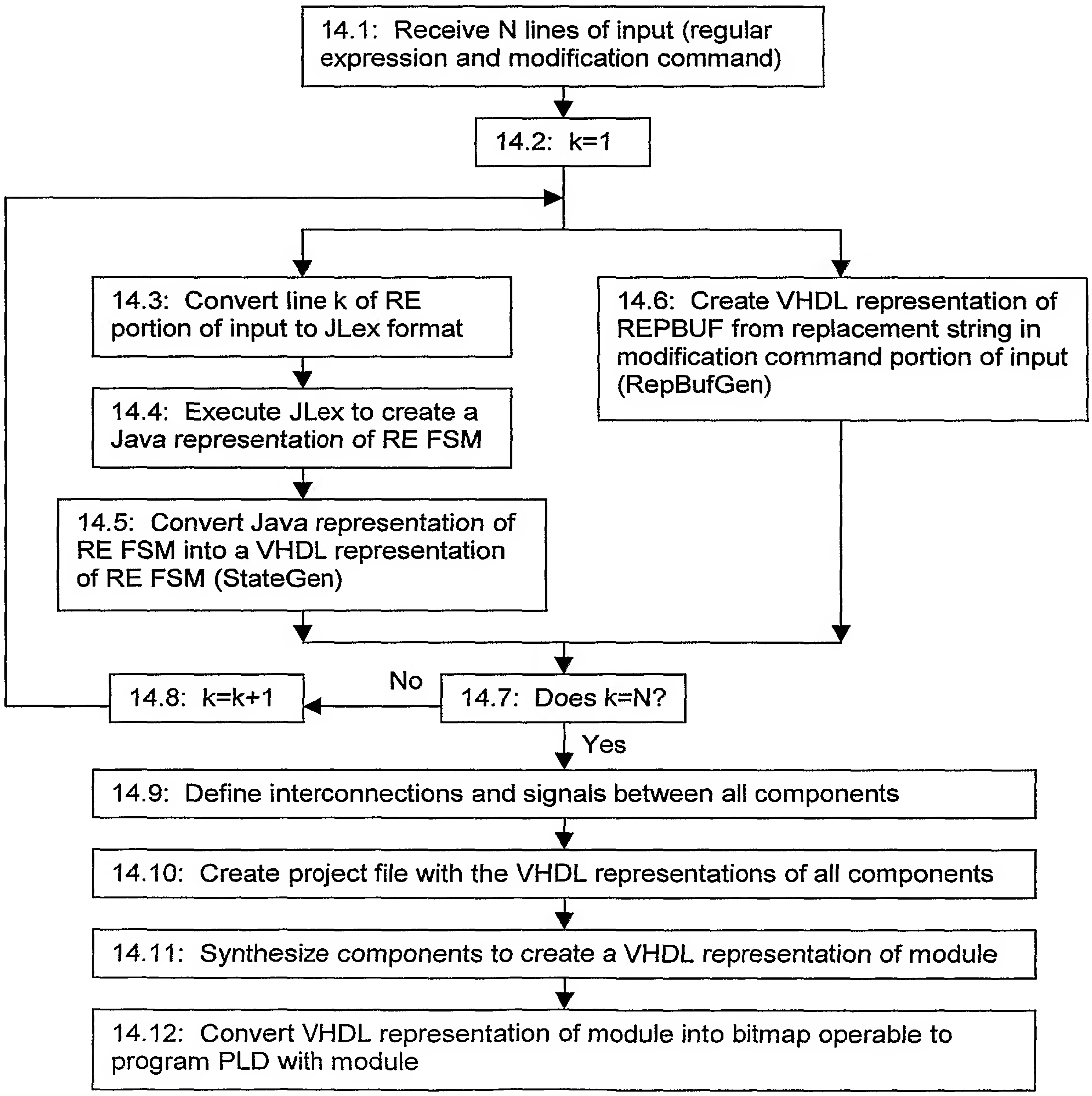
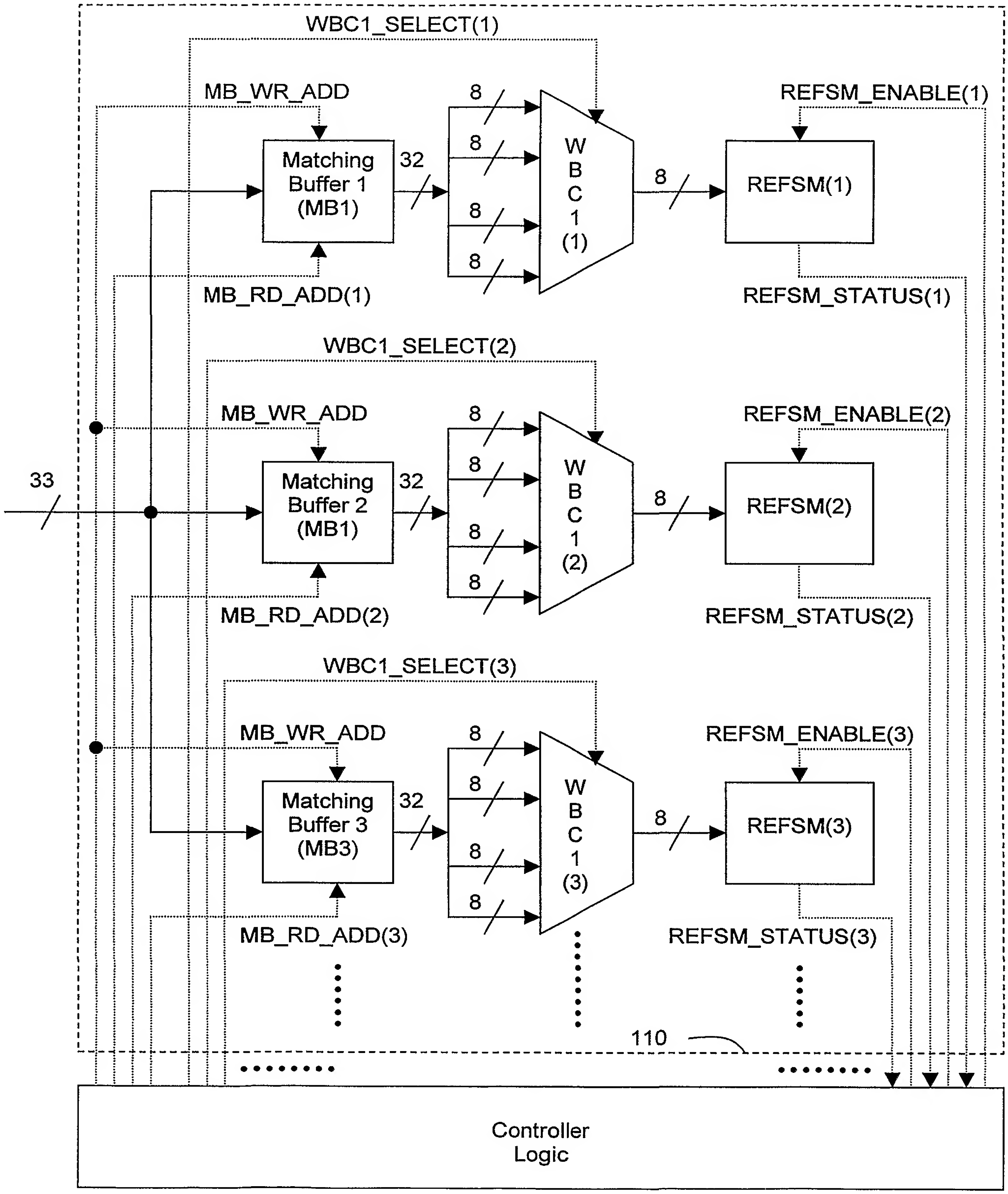


Figure 14



WBC1_SELECT(1)
MB_WR_ADD
8
REFSM_ENABLE(1)
Matching Buffer 1 (MB1)
32
8
W B C 1 (1)
8
REFSM(1)
8
8
MB_RD_ADD(1)
REFSM_STATUS(1)
WBC1_SELECT(2)
MB_WR_ADD
8
REFSM_ENABLE(2)
8
33
Matching Buffer 2 (MB1)
32
W B C 1 (2)
8
REFSM(2)
8
8
MB_RD_ADD(2)
REFSM_STATUS(2)
WBC1_SELECT(3)
MB_WR_ADD
8
REFSM_ENABLE(3)
8
Matching Buffer 3 (MB3)
32
W B C 1 (3)
8
REFSM(3)
8
8
MB_RD_ADD(3)
REFSM_STATUS(3)
110
Controller Logic

Figure 15

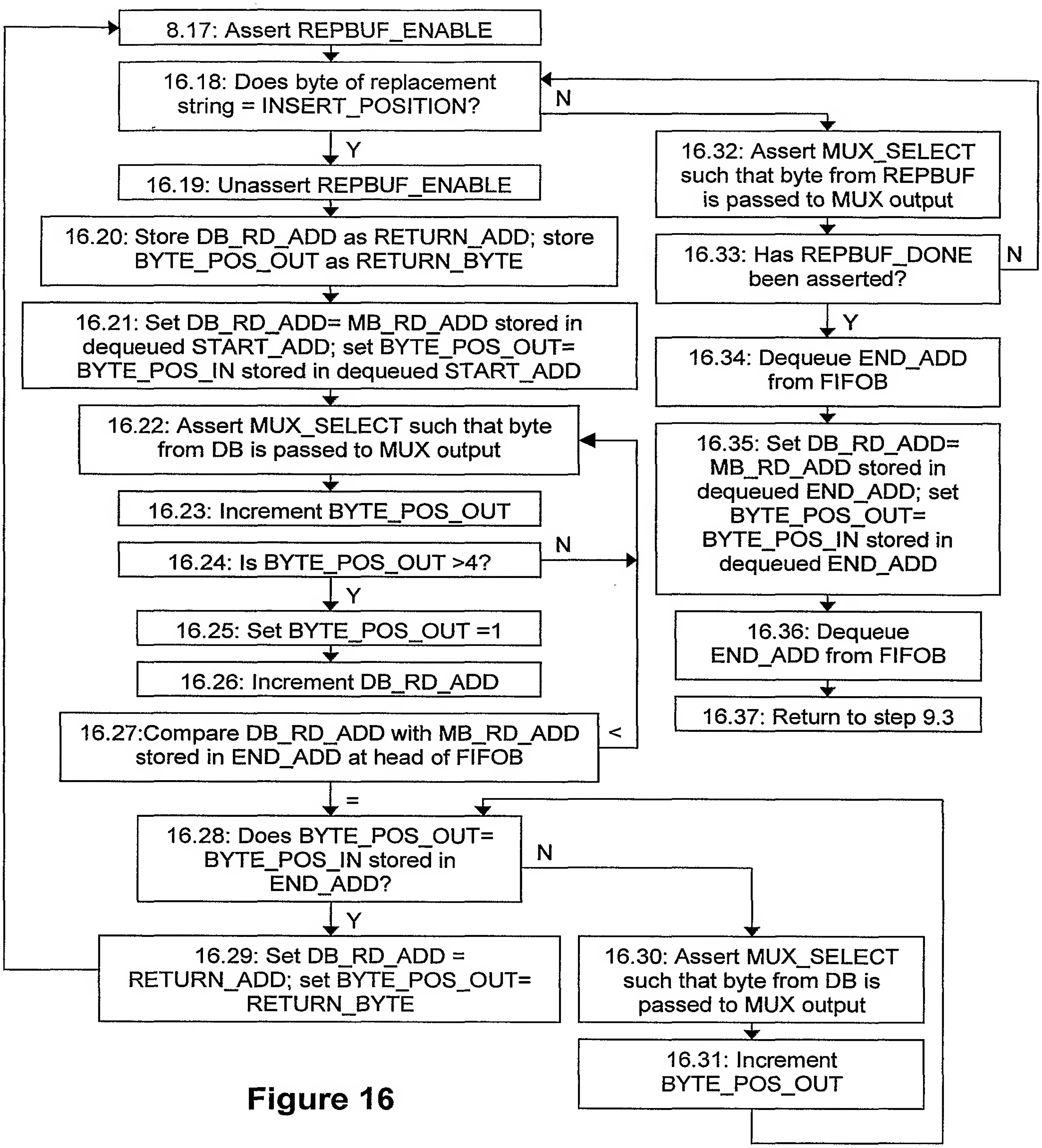

8.17: Assert REPBUF_ENABLE
16.18: Does byte of replacement string = INSERT_POSITION?
N
Y
16.19: Unassert REPBUF_ENABLE
16.20: Store DB_RD_ADD as RETURN_ADD; store BYTE_POS_OUT as RETURN_BYTE
16.21: Set DB_RD_ADD= MB_RD_ADD stored in dequeued START_ADD; set BYTE_POS_OUT= BYTE_POS_IN stored in dequeued START_ADD
16.22: Assert MUX_SELECT such that byte from DB is passed to MUX output
16.23: Increment BYTE_POS_OUT
16.24: Is BYTE_POS_OUT >4?
N
Y
16.25: Set BYTE_POS_OUT =1
16.26: Increment DB_RD_ADD
16.27:Compare DB_RD_ADD with MB_RD_ADD stored in END_ADD at head of FIFOB
<
=
16.28: Does BYTE_POS_OUT= BYTE_POS_IN stored in END_ADD?
N
Y
16.29: Set DB_RD_ADD = RETURN_ADD; set BYTE_POS_OUT= RETURN_BYTE
16.30: Assert MUX_SELECT such that byte from DB is passed to MUX output
16.31: Increment BYTE_POS_OUT
16.32: Assert MUX_SELECT such that byte from REPBUF is passed to MUX output
16.33: Has REPBUF_DONE been asserted?
N
Y
16.34: Dequeue END_ADD from FIFOB
16.35: Set DB_RD_ADD= MB_RD_ADD stored in dequeued END_ADD; set BYTE_POS_OUT= BYTE_POS_IN stored in dequeued END_ADD
16.36: Dequeue END_ADD from FIFOB
16.37: Return to step 9.3

Figure 16

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US03/15910

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 15/177
US CL : 709/246, 231, 221
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 709/208, 221, 231, 246; 704/2, 9; 341/51, 52

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Continuation Sheet

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	LOCKWOOD, et al.; "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)"; February 2001 Entrie document	1-67
Y	US 5,481,735 A (MORTENSEN et al.) 02 JANUARY 1996 Abstract, Column 1, Line 60 through Column 2, Line 34 , Column 4, Line 31 through Column 7, Line 11 through Column 8, Line 44, Column 9, Line 41 through Column 13, Line 30	1-67
Y	US 6,023,760 A (KARTTUNEN) 08 FEBRUARY 2000 Abstract, Column 1, Line 12 through Column 3, Lines 30, Column 5, Line 52 through Column 9, Line 65	1-67
Y	US 6,044,407 A (JONES et al.) 28 MARCH 2000 Abstract, Column 6, Line 41 through Column 8, Line 39	1-67

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 September 2003 (28.09.2003)	23 OCT 2003

Name and mailing address of the ISA/US	Authorized officer
Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 Facsimile No. (703)305-3230	Marc D. Thompson James R. Matthews Telephone No. 703-305-3900

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

PCT/US03/15910

Continuation of Item 4 of the first sheet:
REPROGRAMMABLE HARDWARE FOR EXAMINING NETWORK STREAMING DATA TO DETECT REDEFINABLE PATTERNS AND DEFINE RESPONSIVE PROCESSING

Continuation of B. FIELDS SEARCHED Item 3:
Pat docs: EAST text USPAT, EPO, JPO; NPL: IEEE, ACM
Terms: packet processing, regular expression, snarf, packet/datagram modification, reprogramming PLDs/FPGA, computer network, data streaming